- 数据结构
  - 栈-LIFO
  - 队列-FIFO
    - 普通队列
    - 优先队列
    - 双端队列
  - 链表（感觉OI里用的不多？？）
    - 单向链表
    - 双向(循环)链表
  - 例题

数据结构

栈-LIFO

`stack st`创建栈

`st.top()`返回栈顶元素

`st.push()`传入元素到栈顶

`st.pop()`弹出栈顶

`st.empty()`返回栈是否为空

`st.size()`返回栈里元素数量

`st1=st`把`st`赋值给`st1`

队列-FIFO

普通队列

`queue q`创建队列

`q.front()`返回队首元素

`q.back()`返回队尾元素

`q.push()`在队尾插入元素

q.pop()弹出队首元素

q.empty()返回队列是否为空

q.size()返回队列元素数量

q1=q把q赋值给q1

优先队列

priority_queue<结构类型> 队列名

```
priority_queue <int,vector<int>,greater<int>> p; //从小到大排列(要自己声明)
priority_queue <int,vector<int>,less<int>> p; //从大到小排列(默认)
排序的方式也可以自己写
关键字跟普通的queue一样
```

双端队列

双端队列是指一个可以在队首/队尾插入或删除元素的队列。相当于是栈与队列功能的结合。

deque q创建双端队列

q.front()返回队首元素

q.back()返回队尾元素

q.push_back()在队尾插入元素

q.pop_back()弹出队尾元素

q.push_front()在队首插入元素

q.pop_front()弹出队首元素

q.insert()在指定位置前插入元素

q.erase()删除指定位置前元素

q.empty()返回队列是否为空

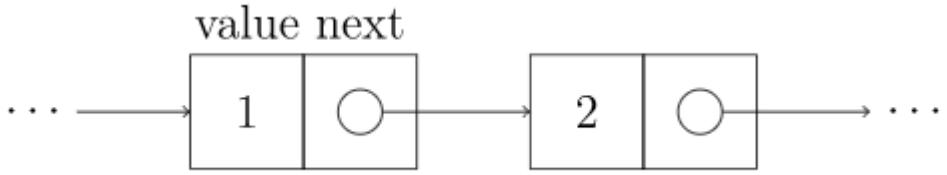
q.size()返回队列元素数量

链表（~~感觉OI里用的不多？？~~）

特点是插入与删除数据十分方便【O(1)】，但寻找与读取数据的表现欠佳【O(n)】

单向链表

单向链表中包含数据域和指针域，其中数据域用于存放数据，指针域用来连接当前结点和下一节点

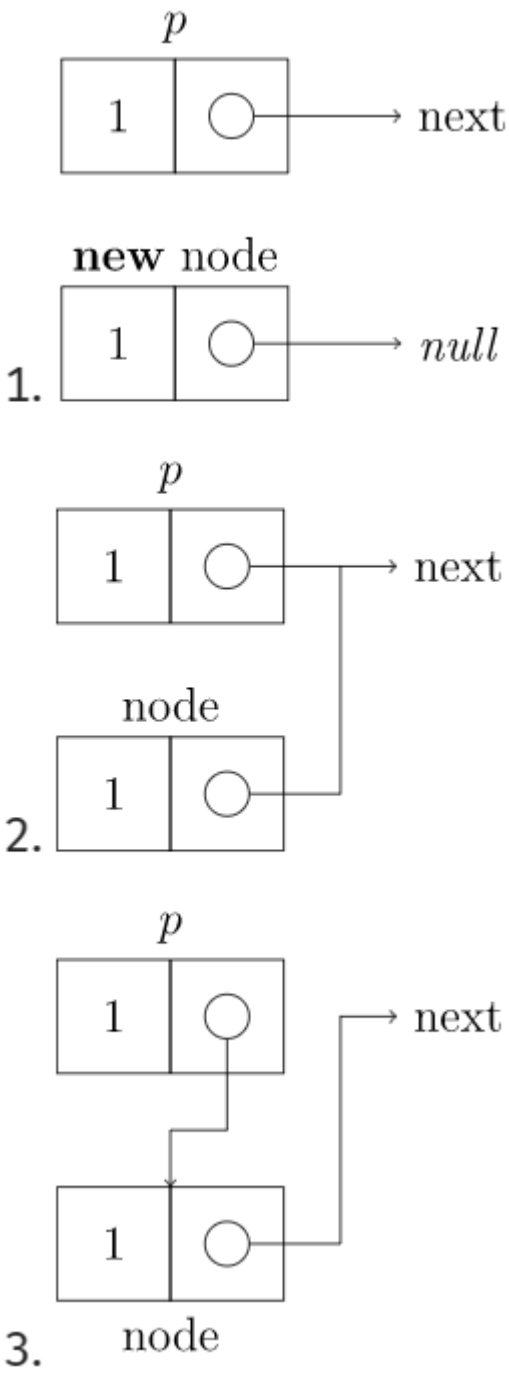


代码：

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

插入数据：

1. 初始化待插入的数据 `node`；
2. 将 `node` 的 `next` 指针指向 `p` 的下一个结点；
3. 将 `p` 的 `next` 指针指向 `node`。



代码：

```
void insertNode(int i.Node *p)
{
    Node *node=new Node;
    node->value=i;
    node->next=p->next;
    p->next=node;
}
```

打印：

```
void printlist(Node*Head)
{
```

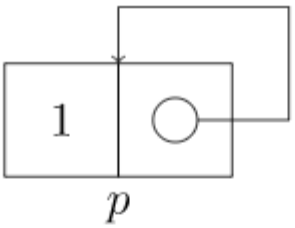
```
    Node *p=Head;
    while(p!=NULL)
    {
        cout<<p->val<<endl;
        p=p->next;
    }
}
```

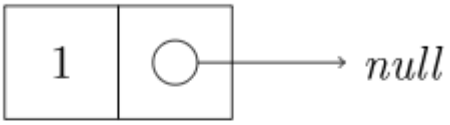
单向循环链表

将链表的头尾连接起来，链表就变成了循环链表。由于链表首尾相连，在插入数据时需要判断原链表是否为空：为空则自身循环，不为空则正常插入数据。

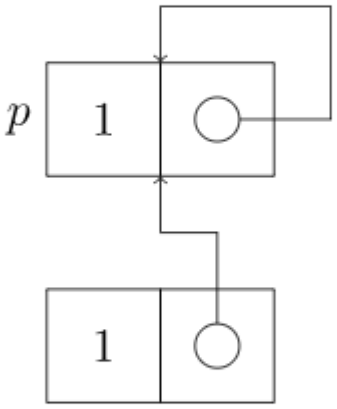
插入数据：

1. 初始化待插入的数据 `node`；
2. 判断给定链表 `p` 是否为空；
3. 若为空，则将 `node` 的 `next` 指针和 `p` 都指向自己；
4. 否则，将 `node` 的 `next` 指针指向 `p` 的下一个结点；
5. 将 `p` 的 `next` 指针指向 `node`。

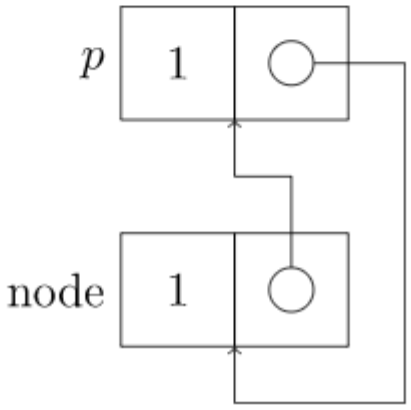




1. **new** node



2.



3.

代码：

```
void insertNode(int i, Node *p)
{
    Node *node = new Node;
    node->value = i;
    node->next = NULL;
    if (p == NULL)
    {
        p = node;
        node->next = node;
    }
    else
    {
```

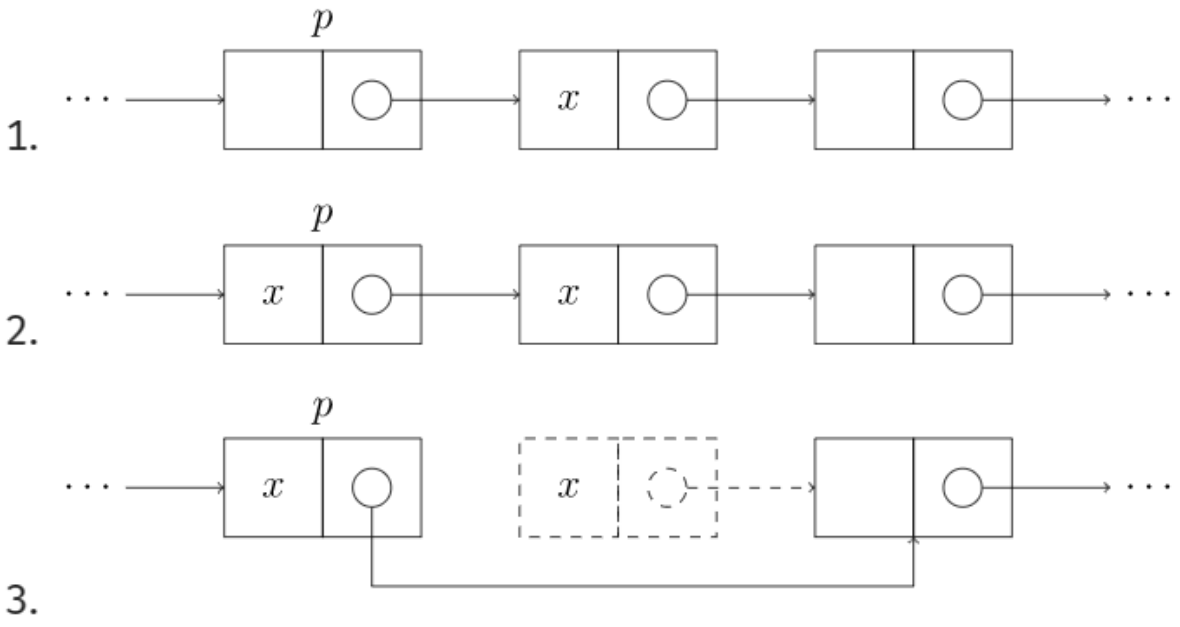
```
        node->next = p->next;
        p->next = node;
    }
}
```

删除数据：

设待删除结点为 `p`，从链表中删除它时，将 `p` 的下一个结点 `p->next` 的值覆盖给 `p` 即可，与此同时更新 `p` 的下下个结点。

流程大致如下：

1. 将 `p` 下一个结点的值赋给 `p`，以抹掉 `p->value`；
2. 新建一个临时结点 `t` 存放 `p->next` 的地址；
3. 将 `p` 的 `next` 指针指向 `p` 的下下个结点，以抹掉 `p->next`；
4. 删除 `t`。此时虽然原结点 `p` 的地址还在使用，删除的是原结点 `p->next` 的地址，但 `p` 的数据被 `p->next` 覆盖，`p` 名存实亡。

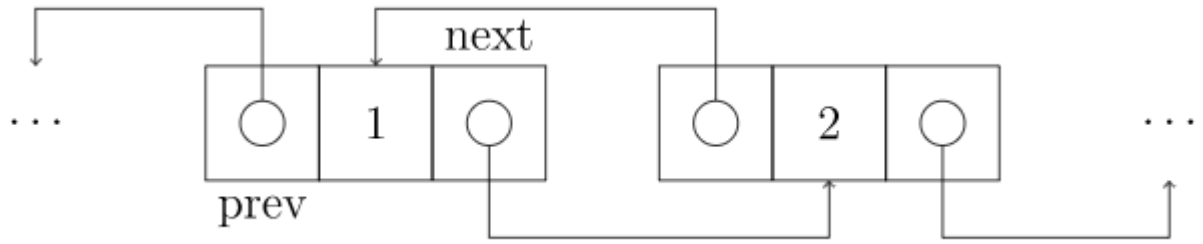


代码：

```
void deleteNode(Node *p)
{
    p->value=p->next->value;
    Node *t=p->next;
    p->next=p->next->next;
    delete t;
}
```

双向(循环)链表

双向链表中同样有数据域和指针域。不同之处在于，指针域有左右（或上一个、下一个）之分，用来连接上一个结点、当前结点、下一个结点。




```
Struct Node
{
    int value;
    Node *left;
    Node *right;
};
```

插入数据：

在向双向循环链表插入数据时，除了要判断给定链表是否为空外，还要同时修改左、右两个指针。

大致流程如下：

1. 初始化待插入的数据 node；
2. 判断给定链表 p 是否为空；
3. 若为空，则将 node 的 left 和 right 指针，以及 p 都指向自己；
4. 否则，将 node 的 left 指针指向 p；
5. 将 node 的 right 指针指向 p 的右结点；
6. 将 p 右结点的 left 指针指向 node；
7. 将 p 的 right 指针指向 node。

代码：

```
void insertNode(int i,Node *p)
{
    Node *node=new Node;
    node->value=i;
    if(p==null)
    {
        p=node;
        node->left=node;
        node->right=node;
    }
    else
```

```
    {
        node->left=p;
        node->right=p->right;
        p->right->left=node;
        p->right=node;
    }
}
```

打印

```
void printlist(Node *first)//传入第一个节点
{
    Node *p;
    for(p=first->right;p;p=p->right)
        cout<<p->value<<endl;
}
```

删除数据

流程大致如下：

1. 将 p 左结点的右指针指向 p 的右节点；
2. 将 p 右结点的左指针指向 p 的左节点；
3. 新建一个临时结点 t 存放 p 的地址；
4. 将 p 的右节点地址赋给 p，以避免 p 变成悬垂指针；
5. 删除 t。

代码:

```
void deleteNode(Node *p)
{
  p->left->right = p->right;
  p->right->left = p->left;
  Node *t = p;
  p = p->right;
  delete t;
}
```

例题

【T1】

P1160 队列安排

提交答案 加入题单

题目描述

复制Markdown 展开

一个学校里老师要将班上 N 个同学排成一列，同学被编号为 $1\sim N$，他采取如下的方法：

1. 先将 1 号同学安排进队列，这时队列中只有他一个人；
2. $2-N$ 号同学依次入列，编号为 i 的同学入列方式为：老师指定编号为 i 的同学站在编号为 $1\sim(i-1)$ 中某位同学（即之前已经入列的同学）的左边或右边；
3. 从队列中去掉 $M(M<N)$ 个同学，其他同学位置顺序不变。

在所有同学按照上述方法队列排列完毕后，老师想知道从左到右所有同学的编号。

输入格式

第 1 行为一个正整数 N，表示了有 N 个同学。

第 $2\sim N$ 行，第 i 行包含两个整数 k,p，其中 k 为小于 i 的正整数，p 为 0 或者 1。若 p 为0，则表示将 i 号同学插入到 k 号同学的左边，p 为 1 则表示插入到右边。

第 $N+1$ 行为一个正整数 M，表示去掉的同学数目。

接下来 M 行，每行一个正整数 x，表示将 x 号同学从队列中移去，如果 x 号同学已经不在队列中则忽略这一条指令。

输出格式

1 行，包含最多 N 个空格隔开的正整数，表示了队列从左到右所有同学的编号，行末换行且无空格。

输入输出样例

输入 #1 复制

```
4
1 0
2 1
1 0
2
3
3
```

输出 #1 复制

```
2 4 1
```

说明/提示

样例解释：

将同学 2 插入至同学 1 左边，此时队列为：

`2 1`

将同学 3 插入至同学 2 右边，此时队列为：

`2 3 1`

将同学 4 插入至同学 1 左边，此时队列为：

`2 3 4 1`

将同学 3 从队列中移出，此时队列为：

`2 4 1`

同学 3 已经不在队列中，忽略最后一条指令

最终队列：

```
2 4 1
```

数据范围

对于 20% 的数据，有 $1 \leq N \leq 10$；

```cpp
#include<bits/stdc++.h>
using namespace std;
const int mx=1e5+10;
struct T//用结构体实现链表
{
    int l,r;

}arr[mx]={0};
void add(int x,int y,int p)//把y同学插到x同学p位置
{
    if(p==1)//插到右边
   {
        arr[y].r=arr[x].r;
        arr[y].l=x;
        arr[x].r=y;
        arr[arr[y].r].l=y;//实际上这里的arr[y].r就是原来的arr[x].r
                          //意思就是原来x右边的现在变成y右边的同学
   }

    else//插到左边
    {
        arr[y].r=x;
        arr[y].l=arr[x].l;
        arr[x].l=y;
        arr[arr[y].l].r=y;//同上
    }
}

void del(int x)//删除第x个人
{
    if(arr[x].r==0&&arr[x].l==0)//如果之前删除过就跳过
    return;
    arr[arr[x].l].r=arr[x].r;//被删的右边现在变成被删的同学左边的同学的右边
    arr[arr[x].r].l=arr[x].l;//同上
    arr[x].l=arr[x].r=0;//清空该同学信息，相当于把他和其他节点链接的信息切断
}
int main()
{
    int n;
    int x,y,p;//把y加到x的p位置
    cin>>n;
    arr[0].r=0,arr[0].l=0;//
    add(0,1,1);//新定义起始点用来初始化
    for(int i=2;i<=n;i++)//因为0.1都被定义好了所以循环要从第三个(实际上是第二个)同学开始
    {
        cin>>x>>p;
        add(x,i,p);
```

```
    }
    int out;//输入要删除元素的个数
    cin>>out;
    for(int i=0;i<out;i++)//删除元素
    {
        int pos;
        cin>>pos;
        del(pos);
    }
    for (int i=arr[0].r;i;i=arr[i].r)//打印(注意循环的条件写法[妙啊])
         cout<<i<<' ';
}
```

【T2】

P1175 表达式的转换

提交答案 加入题单

题目描述

复制Markdown 展开

平常我们书写的表达式称为中缀表达式，因为它将运算符放在两个操作数中间，许多情况下为了确定运算顺序，括号是不可少的，而后缀表达式就不必用括号了。

后缀标记法：书写表达式时采用运算紧跟在两个操作数之后，从而实现了无括号处理和优先级处理，使计算机的处理规则简化为：从左到右顺序完成计算，并用结果取而代之。

例如：`8-(3+2*6)/5+4` 可以写为：`8 3 2 6 * + 5 / - 4 +`

其计算步骤为：

```
8 3 2 6 * + 5 / - 4 +
8 3 12 + 5 / - 4 +
8 15 5 / - 4 +
8 3 - 4 +
5 4 +
9
```

编写一个程序，完成这个转换，要求输出的每一个数据间都留一个空格。

输入格式

就一行，是一个中缀表达式。输入的符号中只有这些基本符号 `0123456789+-*/^()`，并且不会出现形如 `2*-3` 的格式。

表达式中的基本数字也都是一位的，不会出现形如 `12` 形式的数字。

所输入的字符串不要判错。

输出格式

若干个后缀表达式，第 $i+1$ 行比第 i 行少一个运算符和一个操作数，最后一行只有一个数字，表示运算结果。

输入输出样例

输入 #1 复制

```
8-(3+2*6)/5+4
```

输出 #1 复制

```
8 3 2 6 * + 5 / - 4 +
8 3 12 + 5 / - 4 +
8 15 5 / - 4 +
8 3 - 4 +
5 4 +
9
```

输入 #2 复制

```
2^2^3
```

输出 #2 复制

```
2 2 3 ^ ^
2 8 ^
256
```

说明/提示

运算的结果可能为负数，`/` 以整除运算。并且中间每一步都不会超过 2^{31}。字符串长度不超过 100。

注意乘方运算 `^` 是从右向左结合的，即 `2 ^ 2 ^ 3` 为 `2 ^ (2 ^ 3)`，后缀表达式为 `2 2 3 ^ ^`。

其他同优先级的运算是从左向右结合的，即 `4 / 2 / 2 * 2` 为 `((4 / 2) / 2) * 2`，后缀表达式为 `4 2 / 2 / 2 *`。

保证不会出现计算乘方时幂次为负数的情况，故保证一切中间结果为整数。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
const int N=2000;
const long long p=-(1ll<<32+10);//判断这一位是否已运算过
const long long jia=-(1ll<<32+11);//标记判断+
const long long che=-(1ll<<32+12);//标记判断*
const long long chu=-(1ll<<32+13);//标记判断/
const long long ian=-(1ll<<32+14);//标记判断-
const long long fan=-(1ll<<32+15);//标记判断^
char s[N],sta2[N]/*符号栈*/,sta[N]/*表达式栈*/;
int top=0,sta3[N]/*括号栈*/,top3=0,nex[N],len,cnt=0;
long long a[N];
void dfs(int pos,int tp)
{
    int ne=nex[pos],top2=tp;
    if(s[pos]=='(')pos++;//若第一位就是‘（’需pos++
    for(int i=pos;i<=ne-1;i++)
    {
        if(s[i]=='(')dfs(i,top2),i=nex[i];
        if(s[i]<='9'&&s[i]>='0')sta[++top]=s[i];
        if(s[i]=='*'||s[i]=='/'||s[i]=='+'||s[i]=='-'||s[i]=='^')
        {
            if(s[i]=='^')//优先级别考虑
            {
                if(sta2[top2]=='^'&&top2>tp)sta[++top]=sta2[top--];
                sta2[++top2]=s[i];
            }
            if(s[i]=='*'||s[i]=='/')
            {

if((sta2[top2]=='*'||sta2[top2]=='/'||sta2[top2]=='^')&&top2>tp)sta[++top]=sta2[top
2--];
                sta2[++top2]=s[i];
            }
            if(s[i]=='+'||s[i]=='-')
            {
                while(top2>tp)sta[++top]=sta2[top2--];
                sta2[++top2]=s[i];
            }
        }
    }
    while(top2>tp)sta[++top]=sta2[top2--];//在每层dfs要结束是必须把当前层剩余的符号入栈
}
int main()
{
    scanf("%s",s+1);
    len=strlen(s+1);
    for(int i=1;i<=len;i++)//预处理括号位置
    {
        if(s[i]=='(')sta3[++top3]=i;
        if(s[i]==')')nex[sta3[top3--]]=i;
```

```
    }nex[0]=len+1;//因为从第0位开始dfs，所以要让0对应的位置是len+1，注意不能从第1位开始dfs，因为第一位可能是‘（’
    dfs(0,0);
    for(int i=1;i<=top;i++)
    {
        printf("%c ",sta[i]);
        if(sta[i]<='9'&&sta[i]>='0')a[i]=sta[i]-'0';
        if(sta[i]=='-')a[i]=ian;
        if(sta[i]=='*')a[i]=che;
        if(sta[i]=='/')a[i]=chu;
        if(sta[i]=='+')a[i]=jia;
        if(sta[i]=='^')a[i]=fan;
        if(sta[i]=='*'||sta[i]=='/'||sta[i]=='+'||sta[i]=='-'||sta[i]=='^')cnt++;
    }len=top;
    for(int i=1;i<=cnt;i++)
    {
        printf("\n");
        for(int i=1;i<=len;i++)//这层循环是判断当前符号
        {
            if(a[i]==p)continue;
            if(a[i]==jia)
            {
                a[i]=a[i-1]+a[i-2],a[i-1]=p,a[i-2]=p;break;
            }
            if(a[i]==ian)
            {
                a[i]=a[i-2]-a[i-1],a[i-1]=p,a[i-2]=p;break;
            }
            if(a[i]==che)
            {
                a[i]=a[i-1]*a[i-2],a[i-1]=p,a[i-2]=p;break;
            }
            if(a[i]==chu)
            {
                a[i]=a[i-2]/a[i-1],a[i-1]=p,a[i-2]=p;break;
            }
            if(a[i]==fan)
            {
                a[i]=pow(a[i-2],a[i-1]),a[i-1]=p,a[i-2]=p;break;
            }
        }
        int tmp=0;
        for(int i=1;i<=len;i++)//这这层循环是重新处理数组
        {
            if(a[i]==p)continue;
            a[++tmp]=a[i];
        }len=tmp;
        for(int i=1;i<=len;i++)//这里输出当前这一步运算结果
        {
            if(a[i]>p)printf("%lld ",a[i]);
            if(a[i]==jia)printf("+ ");
            if(a[i]==ian)printf("- ");
            if(a[i]==che)printf("* ");
            if(a[i]==chu)printf("/ ");
            if(a[i]==fan)printf("^ ");
        }
    }
```

```
}
```

【T3】

P1241 括号序列

提交答案 加入题单

题目描述

复制Markdown 展开

定义如下规则：

1. 空串是「平衡括号序列」
2. 若字符串 S 是「平衡括号序列」，那么 [S] 和 (S) 也都是「平衡括号序列」
3. 若字符串 A 和 B 都是「平衡括号序列」，那么 AB（两字符串拼接起来）也是「平衡括号序列」。

例如，下面的字符串都是平衡括号序列：

`()`，`[]`，`(())`，`([])`，`()[]`，`()[()]`

而以下几个则不是：

`(`，`[`，`]`，`)(`，`())`，`([()`

现在，给定一个仅由 `(`，`)`，`[`，`]` 构成的字符串 s，请你按照如下的方式给字符串中每个字符配对：

1. 从左到右扫描整个字符串。
2. 对于当前的字符，如果它是一个右括号，考察它与它左侧离它**最近**的**未匹配**的的左括号。如果该括号与之对应（即小括号匹配小括号，中括号匹配中括号），则将二者配对。如果左侧未匹配的左括号不存在或与之不对应，则其配对失败。

配对结束后，对于 s 中全部未配对的括号，请你在其旁边添加一个字符，使得该括号和新加的括号匹配。

输入格式

输入只有一行一个字符串，表示 s。

输出格式

输出一行一个字符串表示你的答案。

输入输出样例

输入 #1 复制

```
([()
```

输出 #1 复制

```
()[]()
```

输入 #2 复制

```
([)
```

输出 #2 复制

```
()[]()
```

说明/提示

数据规模与约定

对于全部的测试点，保证 s 的长度不超过 100，且只含 `(`，`)`，`[`，`]` 四个字符。

```cpp
#include<bits/stdc++.h>
using namespace std;
string str;
int match[110];//用于存放标记是否匹配成功
int main()
{
    cin>>str;
    for(int i=0;i<str.size();i++)
    {
        if(str[i]==')')
        {
            for(int j=i-1;j>=0;j--)
                if(!match[j]&&(str[j]=='['||str[j]=='('))//找到最近的左括号
                {
                    if(str[j]=='('){//匹配的话就把标记置为1
                        match[i]=1;
                        match[j]=1;
                    }
                    break;
                }
        }
        else if(str[i]==']')
        {
            for(int j=i-1;j>=0;j--)
                if(!match[j]&&(str[j]=='['||str[j]=='('))//找到最近的左括号
                {
                    if(str[j]=='['){//匹配的话就把标记置为1
                        match[i]=1;
                        match[j]=1;
                    }
                    break;
                }
        }
    }
     for(int i=0;i<str.size();i++)
        if (match[i])cout<<str[i];//如果有匹配的元素就原样输出
        else if(str[i]=='(')cout<<str[i]<<')';//如果没有匹配的括号就给加上
        else if(str[i]==')')cout<<'('<<str[i];
        else if(str[i]=='[')cout<<str[i]<<']';
        else if(str[i]==']')cout<<'['<<str[i];
        return 0;
}
```

【T4】

P1449 后缀表达式

提交答案 加入题单

题目描述

复制Markdown 展开

所谓后缀表达式是指这样的一个表达式：式中不再引用括号，运算符号放在两个运算对象之后，所有计算按运算符号出现的顺序，严格地由左而右新进行（不用考虑运算符的优先级）。

如：3*(5-2)+7 对应的后缀表达式为：3.5.2.-*7.+@。在该式中，@ 为表达式的结束符号。. 为操作数的结束符号。

输入格式

输入一行一个字符串 s，表示后缀表达式。

输出格式

输出一个整数，表示表达式的值。

输入输出样例

输入 #1 复制

```
3.5.2.-*7.+@
```

输出 #1 复制

```
16
```

说明/提示

数据保证，$1 \leq |s| \leq 50$，答案和计算过程中的每一个值的绝对值不超过 10^9。

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int ans=0;
    stack<int>s;//用栈来解决这个问题
    string str;
    cin>>str;
    for(int i=0;i!='@';i++)
    {
        int num=0;
        int weight=1;//存放位权
        if(str[i]=='.')//遇到这个符号就把前面的字符型数字转化为整型数字
        {
            for(int j=i-1;str[j]>='0'&&str[j]<='9';j--)//注意第二个语句是循环终止条件
            {
                num+=(str[j]-'0')*weight;
```

```
                weight*=10;
            }
            s.push(num);//转换后数入栈
        }
        else if(str[i]>='0'&&str[i]<='9')continue;//如果是数字则跳过
        else//遇到其他的运算符号
        {
            int f=s.top();//先把栈顶的数拿出来用来和下一个数做运算
            s.pop();//栈顶出栈
            if(str[i]=='+')//以下是四则运算
            {
                ans=s.top()+f;
                s.pop();//使用完这个数就让他出栈
            }
            else if(str[i]=='-')
            {
                ans=s.top()-f;
                s.pop();
            }
             else if(str[i]=='*')
            {
                ans=s.top()*f;
                s.pop();
            }
             else if(str[i]=='/')
            {
                ans=s.top()/f;
                s.pop();
            }
            s.push(ans);//把运算之后的结果加到栈里
        }
    }
    cout<<s.top();//只需要输出栈顶（最后肯定只剩一个数）
}
```

【T5】

P4387 【深基15.习9】验证栈序列

提交答案 加入题单

题目描述

复制Markdown 展开

给出两个序列 pushed 和 poped 两个序列，其取值从 1 到 $n(n \leq 100000)$。已知入栈序列是 pushed，如果出栈序列有可能是 poped，则输出 `Yes`，否则输出 `No`。为了防止骗分，每个测试点有多组数据。

输入格式

第一行一个整数 q，询问次数。

接下来 q 个询问，对于每个询问：

第一行一个整数 n 表示序列长度；

第二行 n 个整数表示入栈序列；

第三行 n 个整数表示出栈序列；

输出格式

对于每个询问输出答案。

输入输出样例

输入 #1 复制

```
2
5
1 2 3 4 5
5 4 3 2 1
4
1 2 3 4
2 4 1 3
```

输出 #1 复制

```
Yes
No
```

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=100005;
stack<int>q;
int p,n;
int a[N],b[N];//a为入栈序列，b为出栈序列
int main()
{
    scanf("%d",&p);
    while(p--)
    {
        scanf("%d",&n);
        int cnt=1;//计数器
```

```
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);//初始化数据
        for(int i=1;i<=n;i++)scanf("%d",&b[i]);//
        for(int i=1;i<=n;i++)
        {
            q.push(a[i]);//入栈
            while((q.top())==b[cnt]) //当栈顶元素与b中当前元素相同时出栈
            {
                q.pop();//出栈
                cnt++;//cnt++到b下一个元素
                if(q.empty())break;//结束循环(不然会RE)
            }
        }
        if(q.empty()) cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
        while(!q.empty())q.pop();
    }
    return 0;
}
```