线性表

1. **数组**

STL容器的可变长度数组，头文件是<vector>，有以下的常用方法：

1. vector<int> v(N,i):建立一个可变长度数组v，内部元素类型为int；该可变数组最开始有N个元素，每个元素初始化为i.可以省略i(默认值为0)，也可以把(N,i)同时省略，此时这个数组的长度就是0.内部元素类型可以换成其他的类型，如double.
2. v.push\_back(a):将元素a插入到数组v的末尾，并增加数组长度。
3. v.size():返回数组v的长度。
4. v.resize(n,m):重新调整数组大小为n，如果n比原来的小，则删除多余的信息；如果n比原来的大，则新增的部分都初始化为m，其中m是可以省略的。
5. vector<int>: :iterator it:定义一个名字叫做it的迭代器。
6. v.begin():返回数组v首元素(也就是v[0])的指针(迭代器)。
7. v.end():返回数组v首元素末尾的下一个元素的指针(迭代器)。这个指针有点类似于空指针，不指向任何元素。

除了使用数组下标，还能通过“迭代器”来访问数组中的元素。迭代器有点类似指针(虽然并不能完全画等号)，这里的it就可以认为是一个指向vector中的元素的指针(下文中如果在谈到STL元素的指针，一般都指迭代器)。it可以++或者--变成前一个或后一个元素的指针，也能和指针一样用\*it取该指针中的元素。

由于迭代器和指针在表现方式上很接近，所以v[i]和\*(v.begin()+i)是一样的，都是取对应元素的值。其他STL容器的迭代器也有类似的性质。然而算法竞赛中经常只把vector当作普通的可变数组来使用，比较少用到迭代器。

如果要定义由10个可变数组组成的一个二维数组，可以写为**vector<int> v[10]**.甚至可变数组还能够嵌套，定义一个二维都不定长的二维数组，就像**vector< vector<int> > v** (注意尖括号里的空格，以免会被认为是移位运算符而编译错误)。(**可变多维数组怎么添加元素？**)

1. **栈**

栈的头文件是<stack>，有以下几种方法。

1. stack<int> s:建立一个栈s，其内部元素类型是int
2. s.push(a):将元素a压进栈s
3. s.pop():将s的栈顶元素弹出
4. s.top():查询s的栈顶元素
5. s.size():查询s的元素个数
6. s.empty():查询s是否为空

需要注意的是，这样写虽然方便，但是如果使用STL时不打开-O2优化，就有一点慢。在非常需要追求运行速度的情况下，往往需要自己手写栈。

**括号匹配：**

给定若干字符串，每个字符串由(、)、[、]、{、}这6个字符构成。如果所有的括号都可以匹配上，那么这个字符串合法，否则非法。

1. [({})]合法
2. ()[]{}合法
3. [([]){}[]]{}合法
4. {{}非法
5. ([)(])非法

代码如下：**输入后没输出？**

#include<bits/stdc++.h>  
using namespace std;  
stack<char> s;  
int num;  
char trans(char a){//根据后面的括号找到前面对应的括号   
 if(a==')')return'(';  
 if(a==']')return'[';  
 if(a=='}')return'{';  
 return '\0';  
}  
int main(){  
 cin>>num;  
 string p;  
 getline(cin,p);//假装换行符  
 while(num--){  
 while(!s.empty())s.pop();//清除  
 getline(cin,p);//读入一行  
 for(int i=0;i<p.size();++i){  
 if(s.empty()){//如果栈为空，直接放入栈中   
 s.push(p[i]);  
 continue;  
 }  
 if(trans(p[i])==s.top())  
 s.pop();  
 else s.push(p[i]);  
 }  
 if(s.empty())printf("Yes\n");  
 else printf("No\n");  
 }  
 return 0;  
}

这里遇到了**C++中处理字符串问题的一个坑点**：使用cin读入一个独占的数字后，其读入指针在这一行的末尾，如果再使用getline读入一行字符串时，只会读到空串(第一行)。如果希望读到第二行，则必须要假装读入这一行，可以使用getline，也可以使用getchar等。

**后缀表达式：**

平常写表达式，一般运算符在数的中间，比如1+3\*5，其中+在1和3\*5之间，\*在3和5之间，这种表达式称为中缀表达式。中缀表达式很好用，但是计算机可能没那么喜欢中缀表达式。有一种表达式对计算机很友好，叫作“后缀表达式”。顾名思义，后缀表达式中运算符在参数的后面。

对于计算机而言，后缀表达式当然比中缀表达式更容易计算。另外，后缀表达式可以避免掉括号，这也是它相对于中缀表达式的一大优势。请写一个程序，输入一个长度不超过1000的字符串，求后缀表达式的值。保证后缀表达式中仅有整数和四则运算符号。

例如，输入“2.4.\*1.3.+-@”，其中.是每个数字的结束标志；@是整个表达式的结束标志。输出“4”. 2\*4-(1+3)

**分析：**阅读一个后缀表达式的方法是：从左往右读式子，一旦遇到运算符，就往前取n个数——这个n取决于运算符有多少个参数——然后擦掉这些参数和这个运算符，把计算结果写在那里。接下来，重复刚才的操作，直到表达式中只剩下一个数为止。

代码如下：

#include<bits/stdc++.h>  
using namespace std;  
stack<int> n;  
int s=0,x,y;  
int main(){  
 char ch;  
 do{  
 ch=getchar();  
 if(ch>='0'&&ch<='9')  
 s=s\*10+ch-'0';  
 else if(ch=='.')  
 n.push(s),s=0;  
 else if(ch!='@'){  
 x=n.top();n.pop();y=n.top();n.pop();  
 switch(ch){  
 case'+':n.push(x+y);break;  
 case'-':n.push(y-x);break;  
 case'\*':n.push(x\*y);break;  
 case'/':n.push(y/x);break;  
 }  
 }  
 }while(ch!='@');  
 printf("%d\n",n.top());  
 return 0;  
}

1. **队列**

队列的头文件是<queue>，有以下几种方法。

1. queue<int> q:建立一个队列q，其内部元素类型是int
2. q.push(a):将元素a插入到队列q的末尾
3. q.pop():删除队列q的队首元素
4. q.front():查询q的队首元素
5. q.back():查询q的队尾元素
6. q.size():查询s的元素个数
7. q.empty():查询s是否为空。

如果数据有先进先出的性质，那么可以考虑使用队列；队列常常使用在各类**广度优先搜索**算法上。

1. **链表**

链表的头文件是<list>，有以下几种方法。

1. list<int> a;：定义一个int类型的链表a
2. int arr[5]={1,2,3};list<int> a(arr,arr+3);：从数组arr中的前3个元素作为链表a的初始值
3. a.size()：返回链表的结点数量
4. list<int>::iterator it;：定义一个名为it的迭代器(指针)
5. a.begin();a.end();：链表开始和末尾的迭代器指针
6. it++；it--；:迭代器指向前一个和后一个元素
7. a.push()\_front(x);a.push\_back(x);：在链表开头或者末尾插入元素x
8. a.insert(it,x)：在迭代器it的前插入元素x
9. a.pop\_front();a.pop\_back();：在删除链表开头或者末尾
10. a.erase(it)：删除迭代器it所在的元素
11. for(it=a.begin();it!=a.end();it++)：遍历链表