2018-11-23第六次上机课

第一题 链表+异常处理（50分）

一、 题目描述

1. 实现一个链表，关于特殊情况使用异常进行处理，throw对应的异常即可.实现下面的LinearList类。
2. MyException.h

```
// 异常类型(直接复制到MyException.h, 注意编码):
class MyException{
    public:
        MyException(string s){  emeg  = s;}
        string myMeg(){ return emeg;}
    private:
        string emeg;
};

class OutOfBound:public MyException{//越界异常
    public:
        OutOfBound(string s):MyException(s){}
};

class NullPointer:public MyException{//空指针异常
    public:
        NullPointer(string s):MyException(s){}
};

class ArithmeticeError:public MyException{//算术运算异常
    public:
        ArithmeticeError(string s):MyException(s){}
};

class MemoryLeak : public MyException{//内存泄漏异常
    public:
        MemoryLeak(string s):MyException(s){}
};

class Overflow : public MyException{//溢出异常
    public:
        Overflow(string s):MyException(s){}
}
```

3. LinearList.h

```
//链表结构(直接复制到LinearList.h, 注意编码):
struct Node{
```

```
    Node *next;
    int value;
    Node(int v){ value= v; next = NULL;}
    ~Node(){ next = NULL; }
};
struct List{
    Node* root;
    List(){ root = NULL; }
    ~List(){}
};

//LinearList(复制到LinearList.h,注意编码，其它方法的实现可参照Creation,对于异常,throw即可)
class LinearList{
    public:
        static void Creation(List *&list){//创建一个空的链表
            if(list!=NULL) throw MemoryLeak("MemoryLeak:Creation()");
            list = new List();
        }
        static void Insert(int index,List *list,int value){
            //实现该方法，向list的第index个位置插入value
            //需要考虑的异常情况:list==NULL，越界
        }
        static int Get(int index,List *list){
            //实现该方法，获取list的第index个位置的值并返回
            //需要考虑的异常情况:list==NULL，越界
        }
        static int Remove(int index,List *list){
            //实现该方法，移除list的第index个位置并返回对应的值
            //需要考虑的异常情况:list==NULL，越界
        }
        static int Delete(List *&list){
            //实现该方法，删除list所有元素以及list，并将list置为NULL
            //需要考虑的异常情况:list==NULL
        }
        static void Show(List *list){
            //实现该方法，输出list所有元素,若list不包含元素，则什么都不输出
            //需要考虑的异常情况:list==NULL
            //输出的每一个数字以一个空格隔开，最后的数字后面不包含空格
        }
        static int Add(int index,List *list,int value){
            //实现该方法，向list的第index个元素加上value,并且返回结果
            //需要考虑的异常情况:list==NULL，越界，溢出
        }
        static int Subtract(int index,List *list,int value){
            //实现该方法，向list的第index个元素减去value,并且返回结果
            //需要考虑的异常情况:list==NULL，越界，溢出
        }
        static int Multiply(int index,List *list,int value){
            //实现该方法，向list的第index个元素乘以value,并且返回结果
            //需要考虑的异常情况:list==NULL，越界，溢出
        }
        static int Divide(int index,List *list,int value){
            //实现该方法，向list的第index个元素除以value,并且返回结果
```

```
            //需要考虑的异常情况:list==NULL，越界，算术运算错误
        }
};
```

二、注意事项

1. `list==NULL`与`list`不包含元素是不一样的。
2. `list`的下标从`0`开始，其数值不能为负数。
3. 溢出是指运算结果超过表达的int可表示的范围，算术运算错误是指存在不合法的运算。
    - 例如：假设一台计算机的寄存器都是3位的, 那么能表示的有符号整数位`-4, -3, -2, -1, 0, 1, 2, 3`。现在进行加法运算`2+3=5`,运算结果`5`超过表示范围,则溢出。在计算机中，加减乘除都可能存在溢出的情况。
    - `0x80000000`表示`int`最小值。
    - `0x7FFFFFFF`表示`int`最大值。
4. 对于一个方法的执行，由于参数问题，可能导致多种异常，先检查`NULL`，再检查`Bound`，在检查其它（无需考虑`MemoryLeak`）。
5. 对于输出结果,假设`Insert(1, list, value)`方法出现异常（无需考虑`MemoryLeak`）。
    - 若为`NullPointer`, 则输出`NullPointer:Insert()`
    - 若为`OutOfBound`, 则输出`OutOfBound:Insert(),index=1`
    - 若为`Overflow`或者`Arithmetics`,则输出同上
6. 经过测试发现，本题测试用例正确性***可能***与编译器优化级别有关，故本题g++编译优化级别降低为O1（最低级别优化），若出现在Linux上g++环境编译运行通过所有样例但在oj上出现不通过情况，则应考虑完善自己的代码（不保证Windows下vc++的兼容性，以服务器环境为准）。具体编译命令为`g++ -std=c++0x <in> -fno-asm -Dasm=error -lm -O1 -o <out>`

三、提交要求

1. 将上面所列举的异常类型复制到`MyException.h`文件，`Node`、`List`、`LinearList`的定义和实现均包含在`LinearList.h`文件，直接打包为 zip 格式压缩包，不要存在多一层的目录。
2. 实现代码请严格按照给定的接口名字，否则不能通过编译。
3. 提交代码中不要包含`main()`函数，否则不能通过编译。
4. 严格按照要求的功能实现输出，不要尝试进行其他输入输出活动，否则不能通过测试。

四、测试用例

```
#define C(x) cout << x << endl
```

1. test1
    - code1:

```
List *list = NULL;
try{
    LinearList::Show(list);
    C("Error");
}
catch (OutOfBound &e){
    cout << e.myMeg() << endl;
}
```

```
catch (NullPointer &e){
    cout << e.myMeg() << endl;
    LinearList::Creation(list);
}
catch (ArithmeticeError &e){
    cout << e.myMeg() << endl;
}
catch (MemoryLeak &e){
    cout << e.myMeg() << endl;
}
catch (Overflow &e){
    cout << e.myMeg() << endl;
}

try{
    LinearList::Creation(list);
}
catch (OutOfBound &e){
    cout << e.myMeg() << endl;
}
catch (NullPointer &e){
    cout << e.myMeg() << endl;
}
catch (ArithmeticeError &e){
    cout << e.myMeg() << endl;
}
catch (MemoryLeak &e){
    cout << e.myMeg() << endl;
}
catch (Overflow &e){
    cout << e.myMeg() << endl;
}
if (list != NULL){
    LinearList::Delete(list);
}
```

- output1:

```
NullPointer:Show()
MemoryLeak:Creation()
```

---

2. test2
   - code2:

```
List *list = NULL;
try{
    LinearList::Creation(list);
    LinearList::Show(list);
```

```
    int array[6] = {0, 1, 2, 3, 4, 5};
    for (int i = 0; i < 6; i++){
        LinearList::Insert(i, list, array[i]);
    }
    LinearList::Show(list);
    LinearList::Insert(-1, list, -1);
    LinearList::Delete(list);
}
catch (OutOfBound &e){
    cout << e.myMeg() << endl;
}
catch (NullPointer &e){
    cout << e.myMeg() << endl;
}
catch (ArithmeticeError &e){
    cout << e.myMeg() << endl;
}
catch (MemoryLeak &e){
    cout << e.myMeg() << endl;
}
catch (Overflow &e){
    cout << e.myMeg() << endl;
}
if (list != NULL){
    LinearList::Delete(list);
}
```

- output2:

```
0 1 2 3 4 5
OutOfBound:Insert(),index=-1
```

---

3. test3
   - code3:

```
List *list = NULL;
int array[6] = {0, 1, 2, 3, 4, 5};
while (true){
    try{
        C(LinearList::Get(0, list));
        LinearList::Show(list);
        C(LinearList::Remove(0, list));
        C(LinearList::Get(0, list));
        LinearList::Delete(list);
        break;
    }
    catch (OutOfBound &e){
        cout << e.myMeg() << endl;
```

```
        for (int i = 0; i < 6; i++){
            LinearList::Insert(i, list, array[i]);
        }
    }
    catch (NullPointer &e){
        cout << e.myMeg() << endl;
        LinearList::Creation(list);
    }
    catch (ArithmeticeError &e){
        cout << e.myMeg() << endl;
    }
    catch (MemoryLeak &e){
        cout << e.myMeg() << endl;
    }
    catch (Overflow &e){
        cout << e.myMeg() << endl;
    }
}
```

- output3:

```
NullPointer:Get()
OutOfBound:Get(),index=0
0
0 1 2 3 4 5
0
1
```

---

4. test4
   - code4:

```
List *list = NULL;
int array[6] = {0x7FFFFFFF, 0, 1, 2, 3, 4};
try{
    LinearList::Creation(list);
    for (int i = 0; i < 6; i++){
        LinearList::Insert(i, list, array[i]);
    }
    LinearList::Show(list);
    C(LinearList::Add(1, list, 1));
    LinearList::Show(list);
    LinearList::Add(0, list, 1);
    C("HelloWorld");
    C(LinearList::Add(1, list, 2));
}
catch (OutOfBound &e){
    cout << e.myMeg() << endl;
}
```

```
catch (NullPointer &e){
    cout << e.myMeg() << endl;
}
catch (ArithmeticeError &e){
    cout << e.myMeg() << endl;
}
catch (MemoryLeak &e){
    cout << e.myMeg() << endl;
}
catch (Overflow &e){
    cout << e.myMeg() << endl;
}
try{
    LinearList::Show(list);
    LinearList::Divide(3, list, 0);
}
catch (OutOfBound &e){
    cout << e.myMeg() << endl;
}
catch (NullPointer &e){
    cout << e.myMeg() << endl;
}
catch (ArithmeticeError &e){
    cout << e.myMeg() << endl;
}
catch (MemoryLeak &e){
    cout << e.myMeg() << endl;
}
catch (Overflow &e){
    cout << e.myMeg() << endl;
}
if (list != NULL){
    LinearList::Delete(list);
}
```

output4:

```
2147483647 0 1 2 3 4
1
2147483647 1 1 2 3 4
Overflow:Add(),index=0
2147483647 1 1 2 3 4
ArithmeticeError:Divide(),index=3
```

第二题 I/O编程题（50分）

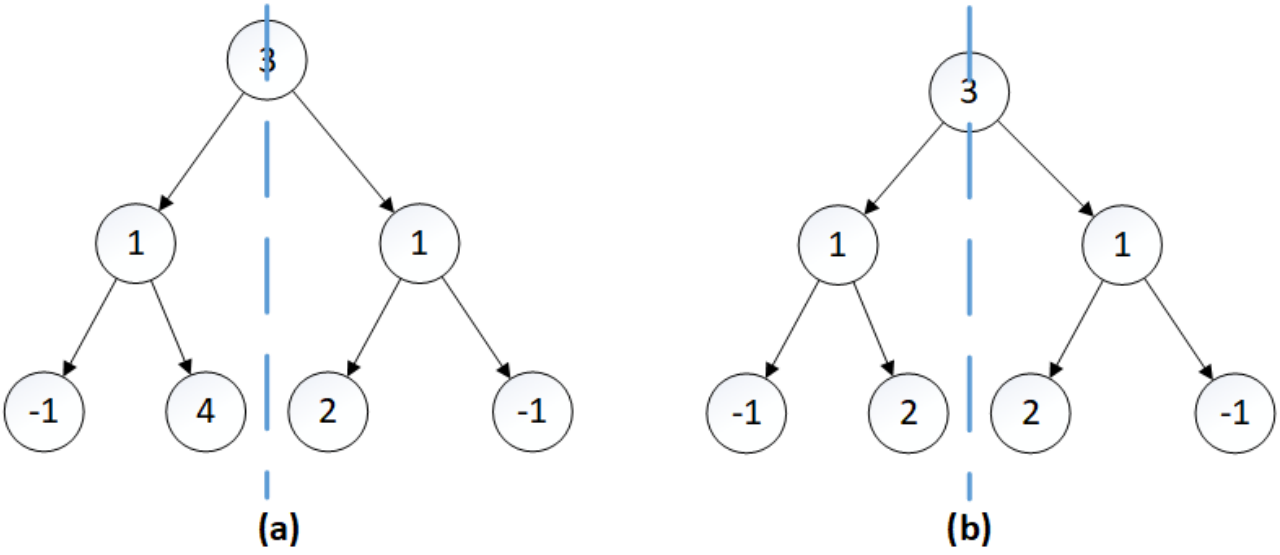
一、题目描述

1. 文件里存放二进制文件，是一串数字，由1~9构成，每个数字占一个int型大小，表示按层遍历的二叉树。编程任务：
   - 判断这个树是否是对称的。
   - 找出树里面最大的数。
   - 已知如果这个树不对称，仅有一对数不相等，按从小到大输出这一对数。
2. SymmetricTree.h

```
class SymmetricTree {
    public:
        vector<int> GetBinaryTree();//用vector装树结构
        bool IsSysmetricTree(); //判断数是否对称
        int GetMaxValue(); //获得数里面节点的最大值
        vector<int> GetDiffValue(); //输出一个大小为2的vector，代表不相等的一对值，按从小到大输出
    private:
        vector<int>tree;//存储树结构
        //根据需要自行添加成员变量
};
```

二、注意事项

1. 数组中树节点值全是正整数。
2. 空节点用-1表示。
3. 文件中有可能省略最后一段的空节点，比如图(a)在文件中可以表示为{3,1,1,-1,4,2,-1}或者{3,1,1,-1,4,2}。再比如假如两个文件中数字分别为{5,-1,-1}和{5}和{5, -1}，实际上他们表示的树一样，都是只有一个数值为5的根节点。



4. 对称表示树以图中蓝色虚线为对称轴，对称轴左边和右边对应的数相同，(a)的第三层2和4不相等，所以(a)不对称，(b)中所有对应位置的数都相等，所有(b)是一个对称的二叉树。
5. 图(a)中最大的数为4，图(b)中最大的数为3。

6. 所有的用例中如果树不对称，则只有一对数不相等，例如图(a)，所以第三个问题输出一个大小为2的vector，vector内容为[2,4]（按从小到大输出）。如果一个树是对称的，则第三个问题输出内容为[0,0]。

三、提交要求

1. 提交一个源码文件SymmetricTree.h，直接打包成zip压缩格式。
2. 实现和声明都写在.h头文件中。
3. 请严格按照给定的接口进行编码，否则无法调用测试用例。
4. 提交的源码文件中，不能包含main()函数，否则无法编译通过。
5. 在代码里不需要任何cout输出，OJ会根据返回值判断是否输出正确。

四、测试用例

1. code:

```
int main() {
    string filename = "filename";
    SymmetricTree *tree = new SymmetricTree();
    tree->GetBinaryTree();
    cout << tree->IsSysmetricTree() << endl;
    cout << tree->GetMaxValue () << endl;
    vector<int> diffValue = tree->GetDiffValue();
    cout << diffValue[0] << "\t" << diffValue[1] << endl;
}
```

2. test1
   - input1(binary file): [3]
   - output1: True 3 [0,0]
3. test2
   - input2(binary file): [3,2]
   - output2: False 3 [-1,2]
4. test3
   - input3(binary file): [4,-1,2]
   - output3: False 4 [-1,2]
5. test4
   - input4(binary file): [3,2,2]
   - output4: True 3 [0,0]
6. test5
   - input5(binary file): [5,2,2,-1,1,3]
   - output5: False 5 [1,3]
7. test6
   - input6(binary file): [6,3,3,-1,2,2,-1]
   - output6: True 6 [0,0]