数据结构课程设计报告



班级：191131班

学号：20131000806

姓名：陈渊

指导老师：

日期：2015/07/13

**题号9 题目：计算命题演算公式的真值**

所谓命题演算公式是指由逻辑变量（其值为TRUE 或FALSE）和逻辑运算符∧（AND）、∨（OR）和┐（NOT）按一定规则所组成的公式（蕴含之类的运算可以用∧、∨和┐来表示）。公式运算的先后顺序为┐、∧、∨，而括号（）可以改变优先次序。已知一个命题演算公式及各变量的值，要求设计一个程序来计算公式的真值。

要求：

（1）程序运行有菜单选择。

（2）利用二叉树来计算公式的真值。首先利用堆栈将中缀形式的公式变为后缀形式；然后根据后缀形式，从叶结点开始构造相应的二叉树；最后按后序遍历该树，求各子树之值，即每到达一个结点，其子树之值已经计算出来，当到达根结点时，求得的值就是公式之真值。

（3）逻辑变元的标识符不限于单字母，而可以是任意长的字母数字串。

（4）根据用户的要求显示表达式的真值表。

1. **需求分析**

1) 由逻辑变量（其值为TRUE或FALSE）【实际操作使用，0代表FALSE，1代表TRUE）】和逻辑运算符∧（AND）、∨（OR）和┐（NOT）【实际操作使用&代表AND,|代表OR，~代表NOT】按一定规则所组成的公式。公式运算的先后顺序为┐、∧、∨，而括号（）可以改变优先次序。输入输出

2) 以人机对话的方式让用户输入要计算的命题表达式；计算出最后的真值并输到屏幕上。

基本思想：

①利用二叉树计算公式的真值:

第一步:利用堆栈将中缀形式的公式变为后缀形式；

第二步:根据后缀形式，从叶结点开始构造相应的二叉树；

第三步:按后序遍历该树，求各子树之值，即每到达一个结点，其子树之值已经计算出来，当到达根结点时，求得的值就是公式之真值；

②逻辑变元的标识符不限于单字母，而可以是任意长的字母数字串；

③根据用户的要求显示表达式的真值表。

**2．设计**

概要设计：

程序入口

等待用户输入

有误

检查输入是否有错误

无误

将中辍表达式转化为后缀表达式

借助后缀表达式构造二叉树

打印二叉树

根据变量真值计算表达式真值

否

是否退出

是

退出程序

**2.1 设计思想**

（1）数据结构设计

包括逻辑结构设计和存储结构设计

用于中辍转换后缀的栈结构：



抽象为模板类：

template<typename Type>class Stack

{

public:

Stack(){};

~Stack(){};

void Push(const Type &item);

Type Pop(void);

Type Top(void)const;

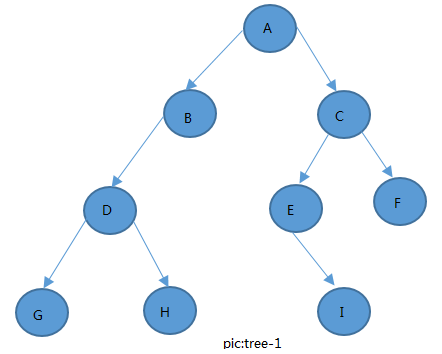
bool IsEmpty(void)const;

private:

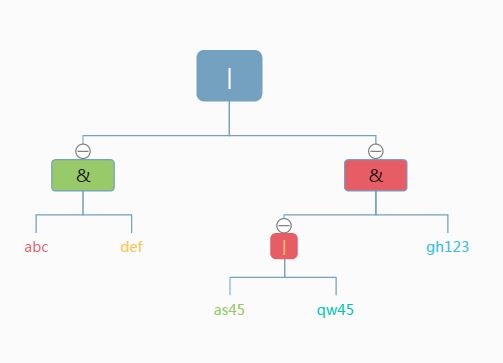
std::vector<Type> m\_Items;

};

用于从后缀表达式计算真值的二叉树



如算式：（abc\*def）|(gh123&(as45|qw45))转化后构造的二叉树如下：



二叉树抽象为：

class Bitree//二叉树类

{

private:

BitreeNode<myData> \*m\_Root;

void Print(BitreeNode<myData> \*node, int level)const;

void Destroy(BitreeNode<myData> \* &node);

public:

Bitree();

~Bitree();

void Visit(BitreeNode<myData> \*node)const;

void Print()const;

void Destroy();

void MakeBitree(myData &data, BitreeNode<myData> \*&left,

BitreeNode<myData> \*&right, BitreeNode<myData> \*&parent,bool sym);

void MakeBitree(BitreeNode<myData> \*&root);

bool IsEmpty()const;

};

（2）算法设计

中缀表达式变成后缀表达式**：**

void Fomula::Transfer();

在将中缀表达式变成后缀表达式的时候会用到堆栈，因此首先需要初始化一个堆栈。又由于逻辑变元可能是字符也可能是字符串，所以它又不同于将单字符的逻辑变元的中缀表达式变成后缀表达式。我的设计是这样的，我将中缀表达式变成后缀表达式的过程分成了两部：化简（将一维的复杂的中缀表达式变成一维的简单的中缀表达式，并将字符串逻辑变元存放在二维数组中），转化（将化简后的中缀表达式变成后缀表达式）。

后缀表达式构造出二叉树**：**

void Fomula::MakeBitree(void)

在这个过程中，我用到了之前所定义的存放树的堆栈。具体实现为：扫描后缀表达式，如果遇到逻辑变元然后将这个变元变成一个树节点，它的实现就是将该逻辑变元赋给树的data域，然后将它的左右子树赋为NULL，然后将这个树节点压入相应的堆栈；接着继续扫描，如果遇到的是单目运算符（非号“！”）也将它构造成一个树节点然后从堆栈里面弹出一个树形节点，将弹出的元素的作为它的左子树，右子树设置为NULL，然后将这个树节点压入相应的堆栈；如果扫描到的是双目运算符（与号“&”或者或号“|”）将它也构造成一棵树，然后将这个树节点压入相应的堆栈，然后从栈中弹出两个元素，一个作为它的左子树，一个作为它的右子树，如此重复n（n为后缀表达式的长度）次。

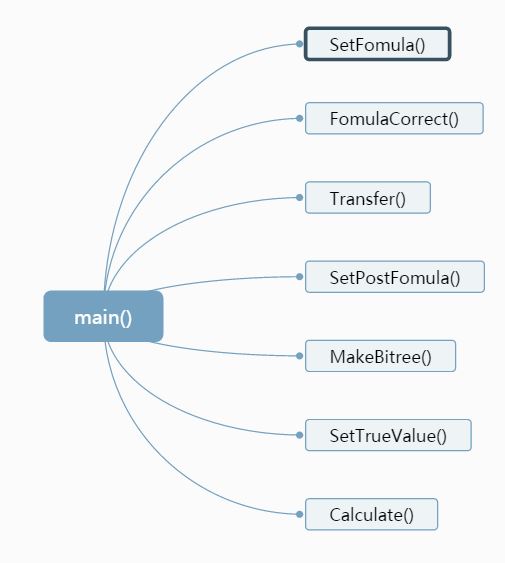
使用中序遍历打印二叉树；

void Bitree::Print(BitreeNode<myData> \*node, int level)const

1. 先查找当前节点的右子树，为空转2.否则递归查找；
2. 打印当前节点信息转3；
3. 查找当前节点的左子树，为空退出，否则递归查找；

**2.2 设计表示**

（1）函数调用关系图



GetItem()

RightChild()

LeftChild()

IsEmpty()

Pop()

Push()

Top()

（2）函数接口规格说明

void SetFomula(string &str);

//初始化Fomula类的使用数据，参数str为用户输入的公式。

void SetPostFomula();

//根据Fomula类内部数据得到后缀表达式，在Transfer(void)之后调用

void SetTrueValue(vector<int> &value, int size);

//用户设定变量真值value为对应变量的0,1值，size为值的个数，

string GetPostFomula()const;

//获取后缀表达式

string GetFomula()const;

//获取用户输入的公式

void MakeBitree(void);

//根据得到的后缀表达式建立二叉树，

void Transfer(void);//中辍表达式转换为后辍表达式

bool FomulaCorrect(void);//检查用户输入的字符串括号是否匹配

void Print()const;//输出Fomula类的所有数据成员的值

bool Calculate()const;//计算逻辑公式的真值

**2.3 详细设计**

栈：

template<typename Type>class Stack

{

public:

Stack(){};

~Stack(){};

//push the item to the top of the vector

void Push(const Type &item);

// pop the top item of vector

Type Pop(void);

//return the top item of the vector

Type Top(void)const;

//judge wether the Stack is empty ,empty return true else return false

bool IsEmpty(void)const;

private:

//the vector of the Stack use for storage the element

std::vector<Type> m\_Items;

};

二叉树：

class Bitree//二叉树类

{

private:

//二叉树头指针

BitreeNode<myData> \*m\_Root;

//输出二叉树，为void Print()const;调用

void Print(BitreeNode<myData> \*node, int level)const;

//删除二叉树，为void Destroy();调用

void Destroy(BitreeNode<myData> \* &node);

public:

Bitree();

~Bitree();

//打印node节点的数据域

void Visit(BitreeNode<myData> \*node)const;

//后序遍历二叉树

void Print()const;

void Destroy();

void MakeBitree(myData &data, BitreeNode<myData> \*&left,

BitreeNode<myData> \*&right, BitreeNode<myData> \*&parent,bool sym);

void MakeBitree(BitreeNode<myData> \*&root);

//判断二叉树是否为空

bool IsEmpty()const;

};

二叉树节点类：

template<typename Type>class BitreeNode//二叉树结点类

{

public:

Type m\_Item; //数据域

BitreeNode<Type> \*m\_Left;//左孩子

BitreeNode<Type> \*m\_Right;//右孩子

BitreeNode<Type> \*m\_Parent;//父节点//为方便使用，添加了指向父节点的指针

//构造函数，

BitreeNode(Type item){};

~BitreeNode(){};

BitreeNode(BitreeNode<Type> \* parent, Type &item){};

BitreeNode( Type &item, BitreeNode<Type> \* left, BitreeNode<Type> \* right,BitreeNode<Type> \* parent){};

BitreeNode<Type>\* &LeftChild(void);

BitreeNode<Type>\* &RightChild(void);

BitreeNode<Type>\* &Parent(void);

void SetItem(Type &item);//设置数据域;

Type GetItem()const;

};

公式操作类;

class Fomula

{

private:

Bitree m\_Tree;//二叉树

string m\_Fomula;//用户输入公式

string m\_postFomula;//后缀表达式

vector<string> m\_varriable;//用于分开存储变量及运算符

string m\_BitreeStr;//二叉树字符串，将整个二叉树数据集合到字符串中

vector<int> m\_TrueValue;//真值栈，存储真值,变量赋值-1

//将数组中的真值插入真值栈

void SetTrueValue(vector<int>&);

//根据构造好的二叉树递归调用计算结果

bool Calculate(BitreeNode<myData> \*ptr)const;

//中序遍历二叉树

void MidVisit(BitreeNode<myData> \*node, int level);

public:

Fomula();

~Fomula();

void SetFomula(string &str);//传入用户输入的公式

void SetPostFomula();

void SetTrueValue(vector<int> &value, int size);//用户设定变量真值

string GetPostFomula()const;

string GetFomula()const;

void MakeBitree(void);//建立二叉树

void Transfer(void);//中辍表达式转换为后辍表达式

bool FomulaCorrect(void);//检查用户输入的字符串括号是否匹配

void Print()const;//输出数据成员的值

bool Calculate()const;//计算公式的真值

void SetTrueValueForTest(vector<bool>&value);

};

**3．调试分析**

调试过程中遇到的问题与解决方案：

1.中缀表达式中的逻辑变元不是单个字符而是一些字符串，这样在将中缀表达式转化成后缀表达式的时候就会比较麻烦，后来经过一番分析与调试后，用二维数组存放比较好，那样实现起来就会比较简单。

2.构造树，在构造树时要用到堆栈，但是前面用到的堆栈的数据类型和此时用到的又有很大的差别，在此时又要想到再换一个类型的堆栈，同时在构造树的时候有要找到合适的算法。

3.真值表的打印，对这一模块的实现最容易想到的就是有几个逻辑变元就进行几次循环，每一重循环对应着一个变量的取值。但是经过分析这显然是行不通的，因为在事先我们并不知道会有多少个变元。最后我用到的方法就是那种最原始的方法，用一重循环去实现，每重循环都会有一个值，将这个值反复进行对2取余和对2进行整除，将取余后的值赋给相应的变元。这样总共循环2的变元素的n次方次即可。

**4．用户手册**

1.用户根据界面提示，输入要计算的逻辑表达式

2.根据提示输入各个变量的真值，0表示false，1表示true

3.用户按照需求查看计算结果，及中间参数

**5．测试数据及测试结果**

****

**6.设计感想：**

计算命题演算公式的真值，这一课题的程序设计的关键点有

1. 中辍表达式转化为后缀表达式，这个算法需要巧妙的运用栈结构Stack的存储特点，发挥思维能力。
2. 利用后缀表达式构造二叉树，这个算法要求设计者熟练运用二叉树的相关知识，再与栈结构相结合。

经过这个课题的设计，我重新找回了课堂上学到的关于栈Stack，和二叉树的相关知识，并且加深了印象。

**7．源程序清单**

**//BitreeNode.h**

#pragma once

#include "stdafx.h"

#include<iostream>

using std::nullptr\_t;

//using namespace std;

template<typename Type>class BitreeNode//二叉树结点类

{

public:

Type m\_Item; //数据域

BitreeNode<Type> \*m\_Left;//左孩子

BitreeNode<Type> \*m\_Right;//右孩子

BitreeNode<Type> \*m\_Parent;//父节点//为方便使用，添加了指向父节点的指针

//构造函数，

BitreeNode(Type item)

:m\_Item(item),m\_Left(NULL),m\_Right(NULL),m\_Parent(NULL){};

~BitreeNode(){};

BitreeNode(BitreeNode<Type> \* parent, Type &item)

:m\_Item(item), m\_Left(NULL), m\_Right(NULL), m\_Parent(parent){};

BitreeNode( Type &item, BitreeNode<Type> \* left, BitreeNode<Type> \* right,BitreeNode<Type> \* parent)

:m\_Item(item), m\_Left(left), m\_Right(right), m\_Parent(parent){};

BitreeNode<Type>\* &LeftChild(void)

{

return m\_Left;

};

BitreeNode<Type>\* &RightChild(void)

{

return m\_Right;

};

BitreeNode<Type>\* &Parent(void)

{

return m\_Parent;

};

void SetItem(Type &item)//设置数据域

{

m\_Item = item;

};

Type GetItem()const//获取数据

{

return m\_Item;

}

};

**///Bitree.h**

#pragma once

#include "stdafx.h"

#include"BitreeNode.h"

#include<string>

using std::string;

typedef string myData;

class Bitree//二叉树类

{

private:

//二叉树头指针

BitreeNode<myData> \*m\_Root;

//输出二叉树，为void Print()const;调用

void Print(BitreeNode<myData> \*node, int level)const;

//删除二叉树，为void Destroy();调用

void Destroy(BitreeNode<myData> \* &node);

public:

Bitree();

~Bitree();

//打印node节点的数据域

void Visit(BitreeNode<myData> \*node)const;

//后序遍历二叉树

void Print()const;

void Destroy();

void MakeBitree(myData &data, BitreeNode<myData> \*&left,

BitreeNode<myData> \*&right, BitreeNode<myData> \*&parent,bool sym);

void MakeBitree(BitreeNode<myData> \*&root);

//判断二叉树是否为空

bool IsEmpty()const;

BitreeNode<myData> \*GetRoot()const;

BitreeNode<myData> \*GetLeft(BitreeNode<myData> \*)const;

BitreeNode<myData> \*GetRight(BitreeNode<myData> \*)const;

};

**//Bitree.cpp**

#include "stdafx.h"

#include "Bitree.h"

#include<iostream>

using namespace std;

Bitree::Bitree() :m\_Root(NULL)

{

}

Bitree::~Bitree()

{

}

bool Bitree::IsEmpty()const

{

if (NULL == m\_Root)return true;

else return false;

}

void Bitree::Visit(BitreeNode<myData> \*node)const

{

cout << node->GetItem();

}

//从node节点开始向下打印二叉树，

void Bitree::Print(BitreeNode<myData> \*node, int level)const

{

if (node != NULL)

{

Print(node->RightChild(), level + 1);//递归调用

if (level != 0)

{

for (int i = 0; i < 6 \* (level - 1); i++)cout << " ";

cout << "----";

}

Visit(node);

cout << endl;

Print(node->LeftChild(), level + 1);

}

}

//打印整棵二叉树

void Bitree::Print(void)const

{

Print(m\_Root, 0);

}

void Bitree::Destroy(BitreeNode<myData> \* &node)

{

if (node != NULL && node->LeftChild() != NULL)

Destroy(node->LeftChild());

if (node != NULL && node->RightChild() != NULL)

Destroy(node->RightChild());

cout << node->GetItem() << " ";

delete node;

}

void Bitree::Destroy()

{

Destroy(m\_Root);

}

void Bitree::MakeBitree(myData &data, BitreeNode<myData> \*&left,

BitreeNode<myData> \*&right, BitreeNode<myData> \*&parent, bool sym)

{

m\_Root = new BitreeNode<myData>(data, left, right, parent);

if(left !=NULL)left->Parent() = m\_Root;

if (right != NULL)right->Parent() = m\_Root;

if (NULL != parent && false == sym)parent->LeftChild() = m\_Root;

else if (NULL != parent && true == sym)parent->LeftChild() = m\_Root;

}

void Bitree::MakeBitree(BitreeNode<myData> \*&root)

{

m\_Root = root;

}

BitreeNode<myData> \* Bitree::GetRoot()const

{

return m\_Root;

}

BitreeNode<myData> \* Bitree::GetLeft(BitreeNode<myData> \*root)const

{

if (NULL!=root)return root->LeftChild();

else return NULL;

}

BitreeNode<myData> \* Bitree::GetRight(BitreeNode<myData> \*root)const

{

if (NULL != root)return root->RightChild();

else return NULL;

}

**//Stack.h**

#ifndef STACK\_H

#define STACK\_H

#include "stdafx.h"

#include<vector>

#include<stdexcept>

template<typename Type>class Stack

{

public:

Stack(){};

~Stack(){};

//push the item to the top of the vector

void Push(const Type &item);

// pop the top item of vector

Type Pop(void);

//return the top item of the vector

Type Top(void)const;

//judge wether the Stack is empty ,empty return true else return false

bool IsEmpty(void)const;

private:

//the vector of the Stack use for storage the element

std::vector<Type> m\_Items;

};

template<typename Type>

void Stack<Type>::Push(const Type &item)

{

m\_Items.push\_back(item);

}

template<typename Type>

Type Stack<Type>::Pop(void)

{

if (m\_Items.empty())

throw std::out\_of\_range("the Stack is empty!");

Type temp = m\_Items.back();

m\_Items.pop\_back();

return temp;

}

template<typename Type>

Type Stack<Type>::Top()const

{

if (m\_Items.empty())

throw std::out\_of\_range("the Stack is empty!");

return m\_Items.back();

}

template<typename Type>

bool Stack<Type>::IsEmpty()const

{

if (m\_Items.empty())return true;

else return false;

}

#endif

**//Fomula.h**

#pragma once

#include "stdafx.h"

#include"Bitree.h"

#include"Stack.h"

#include<vector>

using std::vector;

//typedef int myitem;

class Fomula

{

private:

Bitree m\_Tree;//二叉树

string m\_Fomula;//用户输入公式

string m\_postFomula;//后缀表达式

vector<string> m\_varriable;//用于分开存储变量及运算符

string m\_BitreeStr;//二叉树字符串，将整个二叉树数据集合到字符串中

vector<int> m\_TrueValue;//真值栈，存储真值,变量赋值-1

//将数组中的真值插入真值栈

void SetTrueValue(vector<int>&);

//根据构造好的二叉树递归调用计算结果

bool Calculate(BitreeNode<myData> \*ptr)const;

//中序遍历二叉树

void MidVisit(BitreeNode<myData> \*node, int level);

public:

Fomula();

~Fomula();

void SetFomula(string &str);//传入用户输入的公式

void SetPostFomula();

void SetTrueValue(vector<int> &value, int size);//用户设定变量真值

string GetPostFomula()const;

string GetFomula()const;

void MakeBitree(void);//建立二叉树

void Transfer(void);//中辍表达式转换为后辍表达式

bool FomulaCorrect(void);//检查用户输入的字符串括号是否匹配

void Print()const;//输出数据成员的值

bool Calculate()const;//计算公式的真值

void SetTrueValueForTest(vector<bool>&value);

string GetBitreeStr();

};

//Fomula.cpp

#include "stdafx.h"

#include "Fomula.h"

//#define MYDEBUG

Fomula::Fomula()

{

m\_BitreeStr = " ";

}

Fomula::~Fomula()

{

}

void Fomula::SetFomula(string &str)

{

m\_Fomula = str;

}

void Fomula::SetTrueValue(vector<int> &value)

{

if (!m\_TrueValue.empty())m\_TrueValue.clear();

for (size\_t i = 0; i < value.size(); i++)

m\_TrueValue.push\_back(value[i]);

}

void Fomula::SetTrueValue(vector<int> &Inputvalue,int size)

{

vector<int> value(m\_varriable.size());//临时存储真值

std::cout<<"\n\t请为变量赋值<0:false or 1:true>\n\n";

int num = 0;

for (size\_t i = 0; i < m\_varriable.size(); i++)

{

int flag = 0; size\_t j = 0;

//不是运算符，才赋值

if (m\_varriable[i] != "~"&&m\_varriable[i] != "&"&&m\_varriable[i] != "|")

{

for (j = 0; j < i; j++)

{

if (m\_varriable[i] == m\_varriable[j])

{

flag = 1; break;// /有重复变量时flag为1，且找到第一个重复变量[j]

}

}

if (flag == 0)

{ ///给变量赋真值0或1

int symble = -1;

//std::cout << "\t请设定上述待求表达式中的变量" << m\_varriable[i]<<"的真值(0 or 1?)：\n\t";

if (num >= Inputvalue.size())throw("error !");

symble=Inputvalue[num++];

if (0 == symble)value[i]=0;

else value[i]=1;

}

else

{

value[i] = value[j]; //把重复出现的变量都用第一次的值赋值

}

}

else value[i]=-1;

}

SetTrueValue(value);//将真值压入真值栈

}

string Fomula::GetFomula()const

{

return m\_Fomula;

}

string Fomula::GetPostFomula()const

{

return m\_postFomula;

}

void Fomula::SetPostFomula()

{

m\_postFomula = m\_Fomula+" ";

int n = 0;

for (size\_t i = 0; i < m\_varriable.size(); i++)

{

for (size\_t j = 0; j < m\_varriable[i].size(); j++)

m\_postFomula[n++] = m\_varriable[i][j];

m\_postFomula[n++] = ' ';

}

m\_postFomula = string(m\_postFomula, 0, n);//截尾处理

}

void Fomula::Print()const

{

using std::cout;

using std::endl;

cout << "m\_Fomula:" << m\_Fomula << endl;

cout << "m\_postFomula:" << m\_postFomula << endl;

//while (!m\_Stack.IsEmpty())cout << m\_Stack.Pop() << endl;

for (size\_t i = 0; i < m\_varriable.size(); i++)cout << m\_varriable[i] <<"\t";

cout << endl;

m\_Tree.Print();

}

//判断输入表达式是否有错误

bool Fomula::FomulaCorrect(void)

{

int i = 0;

Stack<char> m\_Stack; //栈

while (m\_Fomula[i]) //表达式没读完时，执行'while'循环

{

#ifdef MYDEBUG

std::cout << m\_Fomula[i];

#endif

//遇到左括号'('将其进栈

if (m\_Fomula[i] == '(')m\_Stack.Push(m\_Fomula[i]);

//栈顶元素为'('且当前元素为')'时，出栈，继续读下面的字符

else if (m\_Fomula[i] == ')'&&!m\_Stack.IsEmpty() && (m\_Stack.Top() == '('))m\_Stack.Pop();

//栈顶元素不为'('且当前元素为')'时，输出'左右括号不匹配'，退出重新输入

else if (m\_Fomula[i] == ')'&&!m\_Stack.IsEmpty() && (m\_Stack.Top() != '('))

return false;

//当前元素为')'但是堆栈已空时候，输出'右括号多于左括号'，退出程序重新输入

else if ((m\_Fomula[i] == ')') && m\_Stack.IsEmpty())

return false;

i++;

}

if (!m\_Stack.IsEmpty()) //此时若堆栈非空，则输出'左括号多于右括号'，退出程序重新输入

return false;

else return true;

}

//将用户输入表达式转化为后缀表达式

void Fomula::Transfer()

{

int k = 0;

Stack<char> m\_Stack; //字符栈

string varriable(100,'0'); //在检查中辍表达式时存储变量

char character; //获取当前栈顶的字符

string tempFomula = m\_Fomula + "#";//加入#符号方便判断算法的结束

size\_t n = tempFomula.size();

m\_Stack.Push('#'); //栈底加入#符号方便判断算法的结束

for (size\_t i = 0; i<n; i++)

{

#ifdef MYDEBUG

std::cout << tempFomula[i];

#endif

//对运算符的解读

if (tempFomula[i] == '~' || tempFomula[i] == '|' || tempFomula[i] == '&' ||

tempFomula[i] == '(' || tempFomula[i] == ')' || tempFomula[i] == '#')

{

while (tempFomula[i] == '~' || tempFomula[i] == '|' || tempFomula[i] == '&' ||

tempFomula[i] == '(' || tempFomula[i] == ')' || tempFomula[i] == '#')

{

character = m\_Stack.Top(); //获取一个栈顶字符

if (character == '#'&&tempFomula[i] == '#')return; //此时堆栈和当前都为‘#’时结束算法

else if

((character == '#'&&tempFomula[i] != '#') || (character == '|'&&tempFomula[i] == '~') || (character == '|'&&tempFomula[i] == '&')

|| (character == '|'&&tempFomula[i] == '(') || (character == '&'&&tempFomula[i] == '~') || (character == '&'&&tempFomula[i] == '(')

|| (character == '~'&&tempFomula[i] == '(') || (character == '('&&tempFomula[i] != ')'))

{

m\_Stack.Push(tempFomula[i]); //当中缀表达式当前运算符优先级较高时，进栈

break;

}

else if (character == '('&&tempFomula[i] == ')')

{

m\_Stack.Pop();//'('和')'相遇时，将'('退栈，接着读下面的

break;

}

else

{

k=0;//将变量开始标志置为0

m\_Stack.Pop(); //当栈顶元素优先级较高时，退栈

varriable[k++] = character;

m\_varriable.push\_back(string(varriable, 0, k));//将字符存入容器

continue;

}

}//while

continue;

}

if (tempFomula[i] != '~' && tempFomula[i] != '|' && tempFomula[i] != '&'

&& tempFomula[i] != '(' && tempFomula[i] != ')' &&tempFomula[i] != '#')

{

k = 0;//表示一个变量的开始

while (tempFomula[i] != '~' && tempFomula[i] != '|' && tempFomula[i] != '&'

&& tempFomula[i] != '(' && tempFomula[i] != ')' &&tempFomula[i] != '#')

{

varriable[k++] = tempFomula[i];

i++;

}

i--;//因为在上面while循环有了i++，for循环也有i++

m\_varriable.push\_back(string(varriable, 0, k));//变量读完，存入容器中

}

}

return ;

}

//从叶节点开始构造二叉树

void Fomula::MakeBitree(void)

{

BitreeNode<myData> \*p, \*q, \*o;

Stack< BitreeNode<myData>\* > tempStack;

size\_t n = m\_varriable.size();

for (size\_t i = 0; i<n; i++)

{

p = new BitreeNode<myData>(m\_varriable[i]); //初始化左右子树以及双亲指针

string str = m\_varriable[i]; //获取变量及运算符存储容器中的一个变量或运算符

if ( "~"== str) //如果是“~”

{

q = tempStack.Pop();

p->m\_Right = q; //将弹出后的结点作为右孩子

q->m\_Parent = p;

tempStack.Push(p);

}

else if ("|"== str || "&"== str)

{

q = tempStack.Pop(); //弹出q作为右孩子

o = tempStack.Pop(); //弹出0作为左孩子

q->m\_Parent= p;

o->m\_Parent= p;

p->m\_Left = o;

p->m\_Right = q;

tempStack.Push( p); //根结点入栈

}

else //如果是变量，直接入栈

{

tempStack.Push(p);

}

}

p= tempStack.Pop();

m\_Tree.MakeBitree(p); //弹出构造好的二叉数的根结点指针，并返回

return ;

}

bool Fomula::Calculate()const

{

if (m\_Tree.IsEmpty())

{

throw ("the Bitree is empty !");

//return false;

}

else

{

BitreeNode<myData> \*ptr = m\_Tree.GetRoot();

return Calculate(ptr);//返回真值，计算结果

}

}

//根据构造好的二叉树计算真值

bool Fomula::Calculate(BitreeNode<myData> \*ptr)const

{

bool leftValue=true, rightValue=true;

if (NULL != ptr->LeftChild())

leftValue = Calculate(ptr->LeftChild());//后序遍历二叉树

if (NULL != ptr->RightChild())

rightValue = Calculate(ptr->RightChild());

if ("~" == ptr->GetItem()) //遇到'~'将值置反

{

if (false == rightValue) return true;

if (true == rightValue) return false;

}

else if ("&" == ptr->GetItem()) //遇到'&'只有两个都为真时才为真

{

if (true == leftValue && true == rightValue) return true;

else return false;

}

else if ("|" == ptr->GetItem()) //遇到'|'只有两个都为假时才为假

{

if (false == leftValue && false == rightValue) return false;

else return true;

}

else

{

//当该结点数据域为变量时

size\_t i = 0;

for (; i < m\_TrueValue.size(); i++)

{

if (m\_varriable[i] == ptr->GetItem())break;

}

if (m\_TrueValue[i]==1)return true; //直接返回该变量的真值

else return false;

}

#ifdef MYDEBUG

std::cout << "Calculate error!" << std::endl;

#endif

return false;

}

void Fomula::SetTrueValueForTest(vector<bool>&value)

{

if (!m\_TrueValue.empty())m\_TrueValue.clear();

for (size\_t i = 0; i < value.size(); i++)

m\_TrueValue.push\_back(value[i]);

}

//中序遍历二叉树，将二叉树集合到string字符串中

void Fomula::MidVisit(BitreeNode<myData> \*node, int level)

{

if (node != NULL)

{

MidVisit(node->RightChild(), level + 1);

if (level != 0)

{

for (int i = 0; i < 6 \* (level - 1); i++)m\_BitreeStr+=" ";

m\_BitreeStr+="---";

}

m\_BitreeStr+= node->GetItem();

m\_BitreeStr+="#";

MidVisit(node->LeftChild(), level + 1);

}

}

string Fomula::GetBitreeStr()

{

MidVisit(m\_Tree.GetRoot(),0);

return m\_BitreeStr;

}

/ MFCFomula.h : PROJECT\_NAME 应用程序的主头文件

#pragma once

#ifndef \_\_AFXWIN\_H\_\_

#error "在包含此文件之前包含“stdafx.h”以生成 PCH 文件"

#endif

#include "resource.h" // 主符号

// CMFCFomulaApp:

// 有关此类的实现，请参阅 MFCFomula.cpp

class CMFCFomulaApp : public CWinApp

{

public:

CMFCFomulaApp();

// 重写

public:

virtual BOOL InitInstance();

DECLARE\_MESSAGE\_MAP()

};

extern CMFCFomulaApp theApp;

// MFCFomula.cpp : 定义应用程序的类行为。

//

#include "stdafx.h"

#include "MFCFomula.h"

#include "MFCFomulaDlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CMFCFomulaApp

BEGIN\_MESSAGE\_MAP(CMFCFomulaApp, CWinApp)

ON\_COMMAND(ID\_HELP, &CWinApp::OnHelp)

END\_MESSAGE\_MAP()

// CMFCFomulaApp 构造

CMFCFomulaApp::CMFCFomulaApp()

{

// 支持重新启动管理器

m\_dwRestartManagerSupportFlags = AFX\_RESTART\_MANAGER\_SUPPORT\_RESTART;

// TODO: 在此处添加构造代码，

// 将所有重要的初始化放置在 InitInstance 中

}

// 唯一的一个 CMFCFomulaApp 对象

CMFCFomulaApp theApp;

// CMFCFomulaApp 初始化

BOOL CMFCFomulaApp::InitInstance()

{

// 如果一个运行在 Windows XP 上的应用程序清单指定要

// 使用 ComCtl32.dll 版本 6 或更高版本来启用可视化方式，

//则需要 InitCommonControlsEx()。 否则，将无法创建窗口。

INITCOMMONCONTROLSEX InitCtrls;

InitCtrls.dwSize = sizeof(InitCtrls);

// 将它设置为包括所有要在应用程序中使用的

// 公共控件类。

InitCtrls.dwICC = ICC\_WIN95\_CLASSES;

InitCommonControlsEx(&InitCtrls);

CWinApp::InitInstance();

AfxEnableControlContainer();

// 创建 shell 管理器，以防对话框包含

// 任何 shell 树视图控件或 shell 列表视图控件。

CShellManager \*pShellManager = new CShellManager;

// 激活“Windows Native”视觉管理器，以便在 MFC 控件中启用主题

CMFCVisualManager::SetDefaultManager(RUNTIME\_CLASS(CMFCVisualManagerWindows));

// 标准初始化

// 如果未使用这些功能并希望减小

// 最终可执行文件的大小，则应移除下列

// 不需要的特定初始化例程

// 更改用于存储设置的注册表项

// TODO: 应适当修改该字符串，

// 例如修改为公司或组织名

SetRegistryKey(\_T("应用程序向导生成的本地应用程序"));

CMFCFomulaDlg dlg;

m\_pMainWnd = &dlg;

INT\_PTR nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: 在此放置处理何时用

// “确定”来关闭对话框的代码

}

else if (nResponse == IDCANCEL)

{

// TODO: 在此放置处理何时用

// “取消”来关闭对话框的代码

}

else if (nResponse == -1)

{

TRACE(traceAppMsg, 0, "警告: 对话框创建失败，应用程序将意外终止。\n");

TRACE(traceAppMsg, 0, "警告: 如果您在对话框上使用 MFC 控件，则无法 #define \_AFX\_NO\_MFC\_CONTROLS\_IN\_DIALOGS。\n");

}

// 删除上面创建的 shell 管理器。

if (pShellManager != NULL)

{

delete pShellManager;

}

// 由于对话框已关闭，所以将返回 FALSE 以便退出应用程序，

// 而不是启动应用程序的消息泵。

return FALSE;

}

// MFCFomulaDlg.h : 头文件

//

#pragma once

#include "afxwin.h"

#include<string>

using namespace std;

// CMFCFomulaDlg 对话框

class CMFCFomulaDlg : public CDialogEx

{

// 构造

public:

CMFCFomulaDlg(CWnd\* pParent = NULL); // 标准构造函数

// 对话框数据

enum { IDD = IDD\_MFCFOMULA\_DIALOG };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

HICON m\_hIcon;

// 生成的消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

public:

afx\_msg void OnBnClickedButtonCalculate();

afx\_msg void OnBnClickedButtonSetValue();

// 用户输入的字符串

CString m\_UserFomula;

// //转化后的后缀表达式

CString m\_PostFomula;

// //用于输出二叉树

CListBox m\_ListBox;

// //计算真值结果

CString m\_TrueValue;

void CMFCFomulaDlg::SetListBox(string str);

// //用户输入的真值

CString m\_EditValue;

afx\_msg void OnEnChangeEditPostfomula();

};

// MFCFomulaDlg.cpp : 实现文件

//

#include "stdafx.h"

#include "MFCFomula.h"

#include "MFCFomulaDlg.h"

#include "afxdialogex.h"

#include"Fomula.h"

#include"SetTrueValue.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

string CStringToString(const CString &cstr)

{

string tempstr(CString::StringLength(cstr), '0');

for (int i = 0; i < CString::StringLength(cstr); i++)

{

tempstr[i] = (char)cstr[i];

}

return tempstr;

}

CString StringToCString(const string &str)

{

CString tempcstr(str.c\_str());

return tempcstr;

}

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialogEx

{

public:

CAboutDlg();

// 对话框数据

enum { IDD = IDD\_ABOUTBOX };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// CMFCFomulaDlg 对话框

CMFCFomulaDlg::CMFCFomulaDlg(CWnd\* pParent /\*=NULL\*/)

: CDialogEx(CMFCFomulaDlg::IDD, pParent)

, m\_UserFomula(\_T(""))

, m\_PostFomula(\_T(""))

, m\_TrueValue(\_T(""))

, m\_EditValue(\_T(""))

{

//设置图标

//m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

m\_hIcon = AfxGetApp()->LoadIcon(IDI\_ICON1);

}

void CMFCFomulaDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

//窗口数据与变量交换

DDX\_Text(pDX, IDC\_EDIT\_Fomula, m\_UserFomula);

DDX\_Text(pDX, IDC\_EDIT\_POSTFOMULA, m\_PostFomula);

DDX\_Control(pDX, IDC\_LIST1, m\_ListBox);

DDX\_Text(pDX, IDC\_EDIT\_VALUE, m\_TrueValue);

DDX\_Text(pDX, IDC\_EDIT1, m\_EditValue);

}

BEGIN\_MESSAGE\_MAP(CMFCFomulaDlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_BUTTON\_CALCULATE, &CMFCFomulaDlg::OnBnClickedButtonCalculate)

ON\_BN\_CLICKED(IDC\_BUTTON\_SET\_VALUE, &CMFCFomulaDlg::OnBnClickedButtonSetValue)

ON\_EN\_CHANGE(IDC\_EDIT\_POSTFOMULA, &CMFCFomulaDlg::OnEnChangeEditPostfomula)

END\_MESSAGE\_MAP()

// CMFCFomulaDlg 消息处理程序

BOOL CMFCFomulaDlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// 将“关于...”菜单项添加到系统菜单中。

// IDM\_ABOUTBOX 必须在系统命令范围内。

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// 设置此对话框的图标。 当应用程序主窗口不是对话框时，框架将自动

// 执行此操作

SetIcon(m\_hIcon, TRUE); // 设置大图标

SetIcon(m\_hIcon, FALSE); // 设置小图标

// TODO: 在此添加额外的初始化代码

return TRUE; // 除非将焦点设置到控件，否则返回 TRUE

}

void CMFCFomulaDlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// 如果向对话框添加最小化按钮，则需要下面的代码

// 来绘制该图标。 对于使用文档/视图模型的 MFC 应用程序，

// 这将由框架自动完成。

void CMFCFomulaDlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // 用于绘制的设备上下文

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// 使图标在工作区矩形中居中

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// 绘制图标

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialogEx::OnPaint();

}

}

//当用户拖动最小化窗口时系统调用此函数取得光标

//显示。

HCURSOR CMFCFomulaDlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

void CMFCFomulaDlg::OnBnClickedButtonCalculate()

{

// TODO: 在此添加控件通知处理程序代码

UpdateData(TRUE);

bool sym = true;//判断用户真值输入是否有问题，有为false

string editvalue = CStringToString(m\_EditValue);

vector<int>value;

if (editvalue.size() < 1)sym = false;

for (int i = 0; i<editvalue.size(); i++)

{

if (editvalue[i] == '0')value.push\_back(0);

else if (editvalue[i] == '1')value.push\_back(1);

else

{

MessageBox(\_T("\t您输入的变量真值有错误，请重新输入！"),

\_T("警告！"), MB\_OKCANCEL | MB\_ICONQUESTION);

sym = false;

}

}

try{

Fomula myFomula;

string sFomula = "~(a123&bcd|ang&cnm&(a123&bcd))";//storage the user's input

sFomula = CStringToString(m\_UserFomula);

//m\_Fomula = StringToCString(sFomula);

myFomula.SetFomula(sFomula);

if (1 <= sFomula.size() && myFomula.FomulaCorrect()&&sym==true)

{

myFomula.Transfer();

myFomula.SetPostFomula();

m\_PostFomula = StringToCString(myFomula.GetPostFomula());

myFomula.MakeBitree();

m\_ListBox.ResetContent();

SetListBox(myFomula.GetBitreeStr());

//myFomula.SetTrueValueForTest(value);//for test

myFomula.SetTrueValue(value, editvalue.size());

if (myFomula.Calculate())m\_TrueValue = "true";

else m\_TrueValue = "false";

}

else

{

MessageBox(\_T("\t您输入的表达式括号不能正确匹配，请重新输入！"), \_T("警告！"), MB\_OKCANCEL | MB\_ICONQUESTION);

}

}

catch (std::exception const &ex)

{

// "exception accure:" << ex.what()

string tempstr(ex.what());

tempstr += "exception accure:";

CString temp = StringToCString(tempstr);

MessageBox(temp, \_T("警告！"), MB\_OKCANCEL | MB\_ICONQUESTION);

}

catch (...)

{

MessageBox(\_T("\t发生未知错误！"), \_T("警告！"), MB\_OKCANCEL | MB\_ICONQUESTION);

}

UpdateData(FALSE);

}

void CMFCFomulaDlg::OnBnClickedButtonSetValue()

{

// TODO: 在此添加控件通知处理程序代码

INT\_PTR nRes; // 用于保存DoModal函数的返回值

CSetTrueValue tipDlg; // 构造对话框类CTipDlg的实例

nRes = tipDlg.DoModal(); // 弹出对话框

if (IDCANCEL == nRes) // 判断对话框退出后返回值是否为IDCANCEL，如果是则return，否则继续向下执行

return;

// 将各控件中的数据保存到相应的变量

UpdateData(TRUE);

// 将被加数和加数的加和赋值给m\_editSum

//m\_editSum = m\_editSummand + m\_editAddend;

// 根据各变量的值更新相应的控件。和的编辑框会显示m\_editSum的值

UpdateData(FALSE);

}

void CMFCFomulaDlg::SetListBox(string str)

{

size\_t lastIndex = 0;

int num = 0;

CString temp;

for (size\_t i = 0; i < str.size(); i++)

{

char tp = str[i];

if (str[i] == '#')

{

string s(str, lastIndex, i-lastIndex);

temp= StringToCString(s);

m\_ListBox.InsertString(num++,temp); // 在列表框中索引为num的位置插入字符串

lastIndex = i + 1;

}

}

//

}

**题号 11 题目：通用试题库软件设计**

题库软件在实际教学测试中得到广泛应用。其内容包括题库文件的选择、题库的建立、试题的查询、修改等操作功能模块，在不同试卷的生成功能中应能根据不同测试水平需求，生成不同难易程度的试卷。

试题库的建立模块应包括试题的录入、修改、删除等。试卷相关参数设置模块应包括试卷难易程度设置、试题量设置、试卷格式设置、试题范围设置等；生成试卷和生成历史试卷模块包括接收相应的参数、打开题库、生成相应试卷、把试卷的全部信息存档；试卷输出模块包括按要求输出相应试卷；题库安全模块要求对题库软件加密并限制访问权限。

【测试数据】

以公共课，如英语或计算机文化基础试题库为例。

**1.需求分析**

1）试题库的使用者分管理员和普通用户，故要区分权限

2）试题库可以添加试题，添加基本库；

3）试题库可以删除试题，删除基本题型库

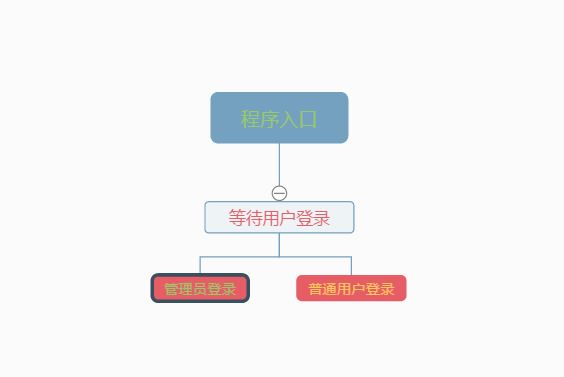
4）用户可以按照不同条件浏览已经存在的试题，试卷，

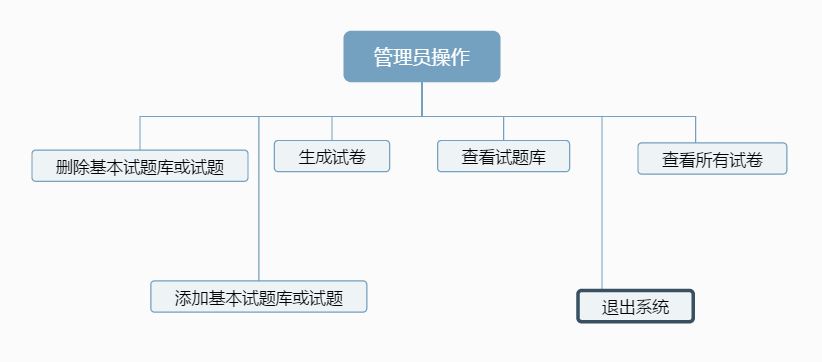
5）生成的试卷应该包含时间作者名称等信息

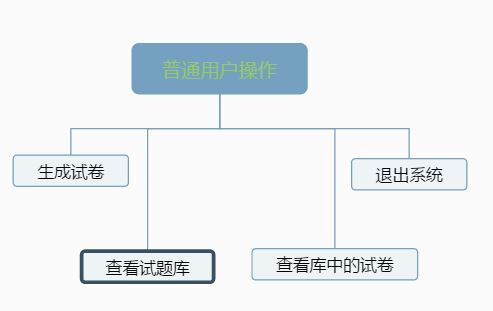
6）添加，删除，修改，生成，之后试题库的信息都要实时保存到文件中

**2．设计**

概要设计：





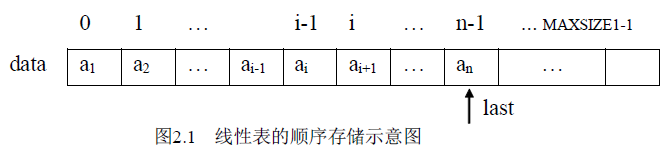


**2.1 设计思想**

（1）数据结构设计（包括逻辑结构设计和存储结构设计）

本试题库本着轻量级操作与用户交互，避免使用复杂的数据存储结构和操作故使用顺序表存储数据；

顺序表可以直接存储数据，支持下标访问。



抽象为SeqList模板类型：

template<typename DataType> class SeqList

{

protected:

DataType \*list; //动态数组

int maxSize; //最大元素个数

int size; //当前元素的个数

void ChangeSize(int Size);//修改顺序表的容量

public:

SeqList(int max = 0); //构造函数

~SeqList(void); //析构函数

SeqList(const SeqList<DataType>&other);

SeqList<DataType> & operator=(const SeqList<DataType> & other);

//获取data在表中的位置，找不到就返回-1；

int GetIndex(const DataType&data);

int Size(void) const; //返回当前表长

void Insert(const DataType& item, int pos); //在位置pos前插元素item

void InsertEnd(const DataType& item); //在list末尾插元素item

DataType Delete(const int pos); //删除位置pos的元素并返回被删元素

DataType &GetData(int pos) ; //返回位置pos的元素

int SetItem(const DataType& item, int pos); //将pos位置上的值改为item

int IsEmpty(void) const; //判断表是否为空

//int Index(DataType& item) const; //定位元素item在顺序表的位置

void Print() const; //输出顺序表

void OrderInsert(DataType& item); //有序插入

void OrderMake(DataType \*d, int len); //构造有序顺序表

void MoreDelete(int low,int high); //删除下标low到high的所有元素，包括low和high

void Merge(const SeqList& La, const SeqList& Lb);//归并有序表La和Lb

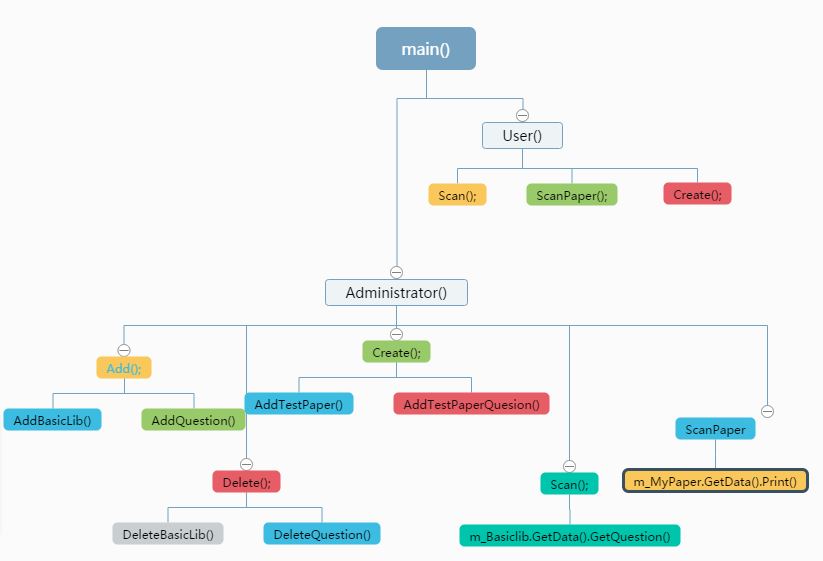
};

（2）算法设计

由于本程序采用顺序表存储数据，故使用的算法不多，有顺序表的顺序查找算法，字符串的关键字查找，比较浅显，不具体介绍。

**2.2 设计表示**

（1）函数调用关系图



（2）函数接口规格说明

void testLib();//管理员登录模块

void userLib();//普通用户登录模块

//添加试题que，type为试题属于的大类，如语文

void AddQuestion(string type, Question &que);

//删除试题，type为大类，index为其在基本库中的索引

void DeleteQuestion(string type, int index);

//获取总库中index位置的试题大类信息

string GetTpye(int index);

//获取总库中类型为type的基本类的位置，找不到就返回-1；

int GetIndex(string type);

////添加基本试题库que，type为试题属于的大类，如语文

void AddBasicLib(string type);

//删除基本试题库试题，type为大类

void DeleteBasicLib(string type);

//根据试题类型type读写文件操作,type 为试题大类，如语文

int ReadFile(string type);//读取文件，并返回其在m\_Basiclib列表中的位置

void WriteFile(string type);

void DeleteFile(string type);//删除名为type的txt文件

bool AlreadyHave(string type);//判断是否已经存在类型为type的基本试题类

void Print(string type);//输出type类型的基本试题库

//在试卷中添加题目

void AddTestPaperQuesion(string papertype, string questionType, int diffi1, int diffi2, int number,string str);

//在容器中添加试卷

void AddTestPaper(string PaperType, string PaperName, string str );

//获取试卷下标

int GetPaperIndex(string PaperName, string str);

//读写试卷信息

void ReadPaper(string PaperName, string str);

void WritePaper(string PaperType, string PaperName, string str );

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void PrintPaper(string PaperType, string PaperName, bool withAnswer, string str);

**2.3 详细设计**

1）因为每道题包含许多数据，抽象为Question数据类型：

class Question/\*题目数据类型抽象\*/

{

public:

Question();

Question(string type, string question, string answer, int diffi);

Question(const Question & other);

Question& operator=(const Question &other);

~Question();

string GetType()const;//获取试题的类型

string GetQuestion()const;//获取试题的描述

string GetAnswer()const;//获取试题答案

int GetDifficulty()const;//获取试题难度系数

void Set(string, string, string, int);//可以修改试题的参数

bool operator<=(const Question &other);//重载的比较操作符，以问题难度m\_Difficulty为对象

bool operator<(const Question &other);

bool operator>=(const Question &other);

bool operator>(const Question &other);

bool operator==(const Question &other);

friend ostream &operator<<(ostream & out, const Question &que);//重载输出符号

private:

string m\_Type;//问题题型type，选择、填空、问答。。

string m\_Question;//问题描述question，

string m\_Answer;//答案answer，

int m\_Difficulty;//问题难度系数difficulty

};

2）试卷这一物件抽象为：

class Paper

{

public:

Paper();

~Paper();

Paper(Paper&other);

string m\_Type;//试卷类型

string m\_Name;//试卷名字

SeqList<string>m\_QuestionType;//试卷试题类型容器

SeqList<int>m\_QuestionSize;//每一种试题的数目

SeqList<Question>m\_Question;//试卷试题容器

void Print(string type, string Qtype);//根据试卷类型type和试题类型Qtype打印试卷信息

void Print(bool answer);

string WriteFile(string type, string Qtype);//根据试卷类型type和试题类型Qtype将试卷信息写入文件

void WriteFile(bool answer);

//为试卷添加试题，

void AddQuestion(BasicLib&arr, string Qtype, int diffi1, int diffi2);

};

3）因为有多种试题类型及试卷类型，故将试题属于的大类抽象为每个试题库：

class BasicLib/\*基本题型库\*/

{

public:

BasicLib();

BasicLib(string );

BasicLib(const BasicLib & other);

~BasicLib();

//获取类型为str的number道题，str 为选择题，填空题，等，diffi2>=难度>=diffi1之间的题目，插尾放入list中

void GetQuestion(SeqList<Question>& list, string type, int diffi1, int diffi2, int number);

//获取类型为str的number道题，str 为选择题，填空题，等，diffi2>=难度>=diffi1之间的题目，插尾放入list中

void GetQuestion(SeqList<Question>& list, string type, int diffi1, int diffi2);

SeqList<Question> & GetSeqList();//获取顺序表的引用，用于顺序表的增加删除修改数据操作

string GetType()const;

void SetType(string str);

//提供关键词搜索功能

//获取类型为含有关键词key的所有类型为type的题，key为关键词，

//难度为插尾放入list中,如果type为空则搜索所有题型,

void GetQuestion(SeqList<Question>& list, int diffi1, int diffi2,string key,string type="" );

//输出基本库中所有数据

void print();

int GetQueTypeSize();//获取m\_QueType

void CalculateQueType();//计算m\_QueType

SeqList<string>m\_QueType;//记录问题类型总个数

private:

string m\_Type;//试题库类型，如语文，英语

SeqList<Question> m\_Que;//顺序表

};

**3．调试分析**

1）在读取文件信息时经常发生读取错位的错误;经分析错误代码如下：

ifile>> difficulty;

ifile.get();

getline(ifile, answer);

因为调用的输入流不一样，当iFile读取之后io缓冲的数据格式与getline要取的数据格式流不一致，故在调用ifile读取数据之后，要在getline调用之前使用ifile.get使io缓冲流恢复正常

2）复制试题操作未成功，经分析，错误代码如下：

Question::Question(Question & other)

{

m\_Type = other.m\_Type;

m\_Question = other.m\_Question;

m\_Answer = other.m\_Answer;

m\_Difficulty = other.m\_Difficulty;

}

这个是试题类型的拷贝构造函数，虽然一般情况下可以赋值成功，但当遇到被赋值对象是特殊修饰时会赋值失败，所以，应将参数改为const 的引用，即Question::Question(const Question & other)；

**4．用户手册**

1）关于输入：用户根据程序运行界面的提示语句选择相应的功能选项或是输入对应的数据即可；在进入程序中的功能选择，一般是输入对应的数字选项；在输入试题信息或类型查找这类的数据就是输入字符串数据。

2）登录界面中管理员登录初始密码是：123456，这个密码可以在当前工程文件key.txt中修改，也可以在登录之后修改。管理员拥有对试题库的所有操作。

3）普通用户登录，不需要密码，普通用户只拥有浏览试题，浏览试卷，生成自己的试卷的权限。

4）试题库初始化时已经自动添加Math，Chinese，English，三种基本试题库，每个库都包含十几道相关的试题，用于测试；

**5．测试数据及测试结果**

**测试数据**

**English：**

选择题

How much is the shirt? A.￡19.15. B.￡9.18. C.￡9.15.

C

1

选择题

What does the woman want to do?A.Find a place.B.Buy a map. C.Get an address.

B

2

选择题

What will the man do for the woman?A. Repair her car.B. Give her a ride.C. Pick up her aunt.

A

3

选择题

Who might Mr.Peterson be?A. A new professor.B. A department head.C. A company director.

A

4

选择题

What does the man think of the book?A. Quite difficult.B. Very interesting.C. Too simple.

C

5

选择题

What are the speakers talking about?A.Weather.B. Clothes.C. News.

B

6

**。。。**

**Math：**

填空题

4.18×0.7的积是（ ）位小数。

23

1

填空题

4.95保留一位小数是（ ）。

14.12

4

填空题

在计算0.73÷0.2时，应先把被除数与除数的小数点都向（ ）移动（ ）位。

124.12

5

填空题

扎一束鲜花需要0.4米丝带,一段长3米的丝带可以扎( )束鲜花。

1.5

3

填空题

当a＝3,b＝1.5时，5.2a－3b＝（ ）。

2.15

5

填空题

用另一种方法表示循环小数0.72727…，是（ ）。

2.135

2

**Chinese：**

选择题

请选出下列词语中加点字的读音完全正确的一项( )(3分)A.惬意(qiè) 匀称(chènɡ) 晨曦(xī) 潜心贯注(qiǎn)B.荣膺(yīnɡ) 愧怍(zhuò) 荒谬(miào) 广袤无垠(mào)C.干涸(hé) 和煦(xù) 栈桥(zhàn) 怒不可遏(è)D.馈赠(zènɡ) 吊唁(yán) 绥靖(suí) 气息奄奄(yān)

C

1

选择题

请选出下列句子中加点成语运用错误的一项( )(3分)A.同学们登上莲花山，仰望着邓小平塑像，回想起小平爷爷一生的丰功伟绩，不禁肃然起敬。B.前不久，某游客在埃及古神庙的神像上刻下“XXX到此一游”几个字，这让在场的其他游客叹为观止，纷纷摇头。C.峰环水抱的萨尔茨堡，高高低低的房屋鳞次栉比，庄严肃穆的修道院坐落在绿树浓阴中。D.从埃文斯语无伦次的话里，斯科特等人终于明白，这个苦命的人已经疯了。

B

2

选择题

请选出下列句子中没有语病的一项( )(3分)A.余华的新书《第七天》即将出版的消息引起了广大读者的关注。B.机动车尾气污染能否治理好是深圳大气环境改善的关键。C.最近，深圳交警在全市范围内开展了“礼让斑马线，文明我先行”，城市的交通状况大为改善。D.深圳长青诗社大约20名左右诗友到大鹏老干部活动中心参观楹联书画展。

B

3

选择题

请选出下列说法错误的一项( )(3分)A.《红楼梦》是我国古代小说的顶峰之作。小说以贾宝玉、林黛玉的爱情悲剧为线索，描写了以贾家为代表的四大家族的兴衰史，反映了封建社会晚期广阔的社会现实。作曹雪芹，清代小说家。B.古代刻在器物上用来警戒自己或者称述功德的文字，叫“铭”，后来成为一种文体。这种文体一般都是用韵的，如刘禹锡的《陋室铭》。C.语气强烈的反问句末尾，也可以用叹号。例如：我哪里比得上他呀！D.“晶莹的露珠”是主谓短语，而“露珠晶莹”是偏正短语。

C

4

填空题

问渠那得清如许？\_\_\_\_\_\_\_\_\_\_\_\_\_。(朱熹《观书有感》)

为有源头活水来

2

填空题

\_\_\_\_\_\_\_\_\_\_\_\_\_，随风直到夜郎西。(李白《闻王昌龄左迁龙标遥有此寄》)

我寄愁心与明月

3

填空题

江山代有才人出\_\_\_\_\_\_\_\_\_\_\_\_\_。(赵翼《论诗》)

各领风骚数百年

4

填空题

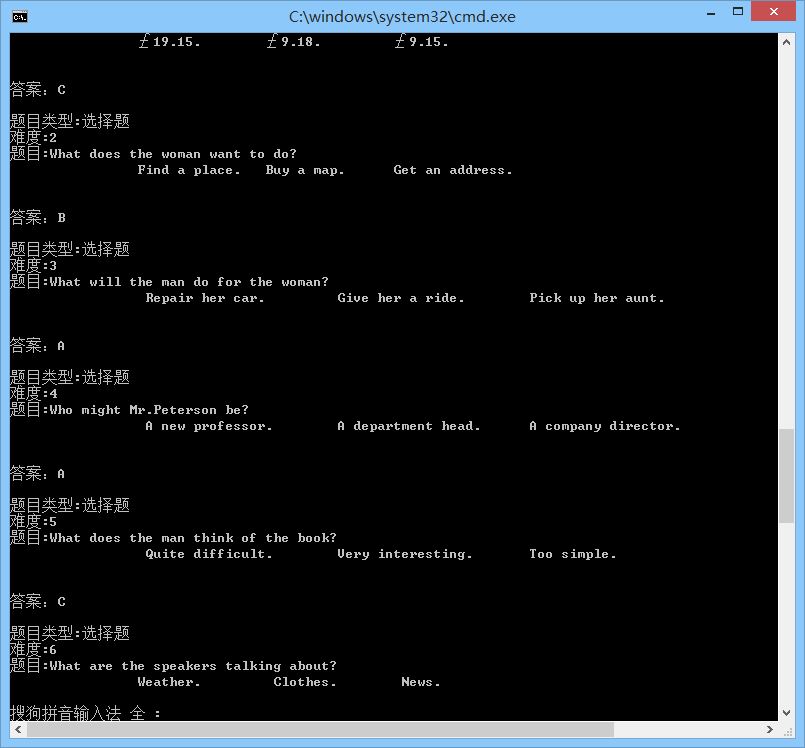
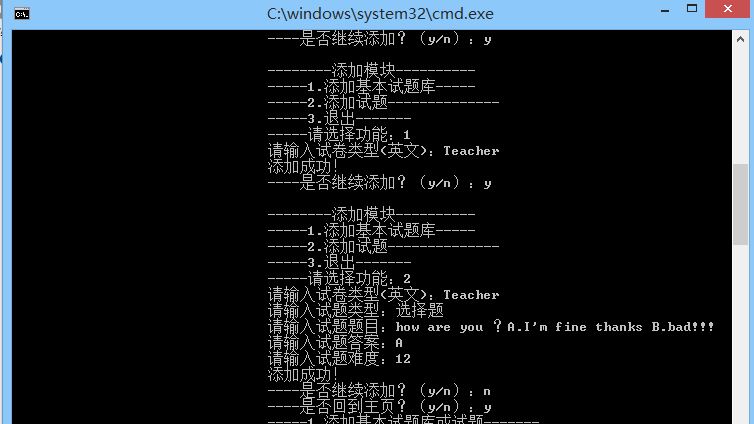
\_\_\_\_\_\_\_\_\_\_\_\_\_，凭君传语报平安。(岑参《逢入京使》)

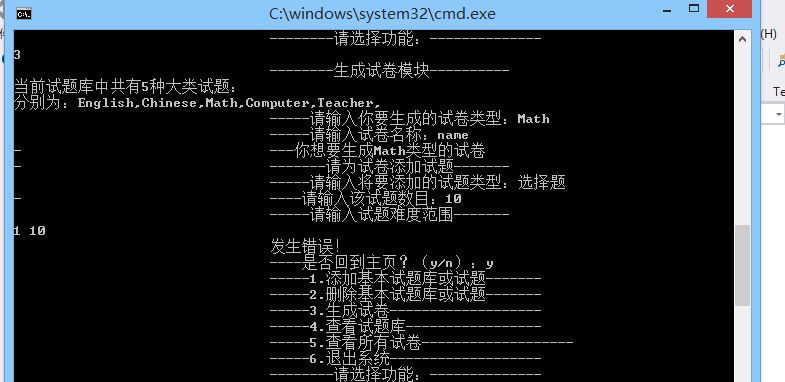
马上相逢无纸笔

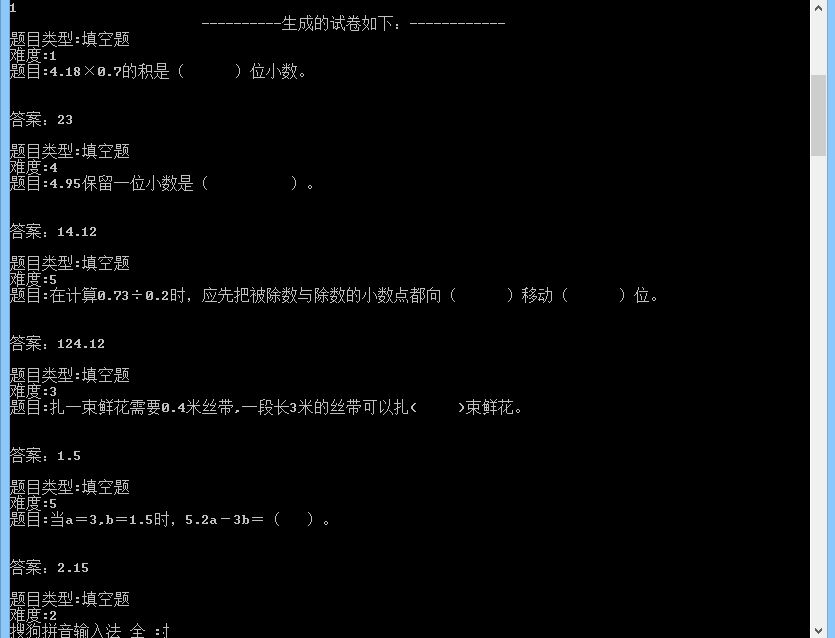
5

**测试结果：**

****

****

****

****

**6.设计感想**

本试题库的设计关键点在于各个数据类型的组织与联系，因为试题库本身要容纳的数据是庞大的，操作是复杂的，数据间必须保证同步更新，每个模块的功能也必须明确出来。经过这个课题的程序设计，极大的锻炼了我的逻辑组织能力，考验了我对于系统开发的架构设计组织。虽然我的能力有限，现在设计的程序还存在许多问题和缺陷，但经过这次设计，我的进步还是十分大的。

**7.附录程序清单**

//SeqList.h

#pragma once

#include <iostream>

using namespace std;

#include <stdlib.h>

#include"Question.h"

template<typename DataType> class SeqList

{

protected:

DataType \*list; //动态数组

int maxSize; //最大元素个数

int size; //当前元素的个数

void ChangeSize(int Size);//修改顺序表的容量

public:

SeqList(int max = 0); //构造函数

~SeqList(void); //析构函数

SeqList(const SeqList<DataType>&other);

SeqList<DataType> & operator=(const SeqList<DataType> & other);

//获取data在表中的位置，找不到就返回-1；

int GetIndex(const DataType&data);

int Size(void) const; //返回当前表长

void Insert(const DataType& item, int pos); //在位置pos前插元素item

void InsertEnd(const DataType& item); //在list末尾插元素item

DataType Delete(const int pos); //删除位置pos的元素并返回被删元素

DataType &GetData(int pos) ; //返回位置pos的元素

int SetItem(const DataType& item, int pos); //将pos位置上的值改为item

int IsEmpty(void) const; //判断表是否为空

//int Index(DataType& item) const; //定位元素item在顺序表的位置

void Print() const; //输出顺序表

void OrderInsert(DataType& item); //有序插入

void OrderMake(DataType \*d, int len); //构造有序顺序表

void MoreDelete(int low,int high); //删除下标low到high的所有元素，包括low和high

void Merge(const SeqList& La, const SeqList& Lb);//归并有序表La和Lb

};

/\*\*

\* 构造函数。置顺序表的当前表长size为0

\*\*/

template<typename DataType>SeqList<DataType>::SeqList(int max)

{

maxSize = max;

size = 0;

list = new DataType[maxSize];

}

/\*\*

\* 析构函数

\*\*/

template<typename DataType>SeqList<DataType>::~SeqList(void)

{

delete[]list;

}

/\*\*

\* 返回顺序表的当前表长size的值

\*\*/

template<typename DataType>int SeqList<DataType>::Size(void) const

{

return size;

}

/\*\*

\* 在指定位置pos前插一个数据元素item

\*\*/

template<typename DataType>void SeqList<DataType>::Insert(const DataType& item, int pos)

{

int i;

if (size == maxSize)

{

/\*cout << "顺序表已满无法插入！" << endl;

exit(0);\*/

if (maxSize == 0||maxSize==1)maxSize++;

ChangeSize( (int)(maxSize\*1.5) );//增加容量

}

if (pos < 0 || pos > size) //当pos等于size时表示插入在最后

{

cout << "Insert 参数pos越界出错!!!" << endl;

exit(0);

}

//从size-1至pos-1逐个元素后移（把前一个元素迁移到后一个元素位置）

for (i = size; i > pos; i--)

list[i] = list[i - 1];

list[pos] = item; //在pos位置插入item

size++; //当前表长size加1

}

/\*\*

\* 在list末尾插一个数据元素item

\*\*/

template<typename DataType>void SeqList<DataType>::InsertEnd(const DataType& item)

{

if (size == maxSize)

{

/\*cout << "顺序表已满无法插入！" << endl;

exit(0);\*/

if (maxSize == 0 || maxSize == 1)maxSize++;

ChangeSize((int)(maxSize\*1.5));//增加1.5倍容量

}

//从size-1至pos-1逐个元素后移（把前一个元素迁移到后一个元素位置）

list[size] = item; //在pos位置插入item

size++; //当前表长size加1

}

/\*\*

\* 删除指定位置pos上的数据元素

\*\*/

template<typename DataType>DataType SeqList<DataType>::Delete(const int pos)

{

if (size == 0)

{

cout << "顺序表已空无元素可删！" << endl;

exit(0);

}

if (pos < 0 || pos > size - 1) //删除元素下标必须在0至size-1之间

{

cout << "参数pos越界出错!!" << endl;

exit(0);

}

DataType temp = list[pos];

//从pos至size-2逐个元素前移（把后一个元素迁移到前一个元素位置，list[size-1]移入list[size-2]中）

for (int i = pos; i < size - 1; i++)

list[i] = list[i + 1];

size--; //当前表长size减1

return temp;

}

/\*\*

\* 取顺序表中位置pos上的元素，参数出错时退出

\*\*/

template<typename DataType>DataType &SeqList<DataType>::GetData(int pos)

{

if (pos < 0 || pos > size - 1) //取的元素序号必须在0至size-1之间

{

cout << "GetData 参数pos越界出错!" << endl;

exit(0); //参数1表示出错退出

}

return list[pos];

}

/\*\*

\* 将pos位置上的值改为item

\*\*/

template<typename DataType>int SeqList<DataType>::SetItem(const DataType& item, int pos)

{

if (pos > size - 1)

return 0;

else

{

list[pos] = item;

return 1;

}

}

/\*\*

\* 判顺序表空否，为空返回1；不空返回0

\*\*/

template<typename DataType>int SeqList<DataType>::IsEmpty(void) const

{

if (size == 0)

return 1;

else

return 0;

}

/\*\*

\* 输出顺序表

\*\*/

template<typename DataType>void SeqList<DataType>::Print()const

{

int i;

for (i = 0; i<size; i++)

{

cout << list[i] << " ";

}

cout << endl;

}

/\*\*

\* OrderInsert()在顺序表L中的合适位置插入数据元素item，使得L中的数据元素按值非递减有序排列。

\*\*/

template<typename DataType>void SeqList<DataType>::OrderInsert(DataType& item)

{

int i = 0;

if (IsEmpty()){ list[0] = item; size++; }

else if (size == maxSize)

{

cout << "顺序表已满无法插入！" << endl;

exit(0);

}

else if (item >= list[size - 1]){ list[size] = item; size++; }

else

for (i = 0; i < size; i++)

{

if (item <= list[i])

{

/\*for (int j = size; j > i; j--)//方法一：i到size-1的元素都后移一位

list[j] = list[j - 1];

list[i] = item;//插入i之后的i+1位置

size++;

break;\*/

Insert(item, i); //方法二：

break; //调用insert()

}

}

}

/\*\*

\* MoreDelete()在顺序表L中删除所有值为item的元素。

\*\*/

template<typename DataType>void SeqList<DataType>::MoreDelete(int low, int high)

{

//删除low到high之间的所有元素

if (low>=0&&high<size)

for (int i = low; i<=high; i++)

{

Delete(low);

}

}

/\*\*

\* OrderMake()从数组d依次取len个数据元素有序插入到有序顺序表L中，构造有序顺序表。

\*\*/

template<typename DataType>void SeqList<DataType>::OrderMake(DataType \*d, int len)

{

for (int i = 0; i<len; i++)

OrderInsert(d[i]);

}

/\*\*

\* 将两个非递减有序的顺序表La和Lb合并为表\*this，并且顺序表\*this也按值非递减有序

\*\*/

template<typename DataType>void SeqList<DataType>::Merge(const SeqList& La, const SeqList& Lb)

{

int i, j;

if (La.Size() + Lb.Size() > maxSize)

{

cout << "数组空间不够，无法合并!";

exit(0);

}

i = 0;

j = 0;

while (i < La.Size() && j < Lb.Size()) /\*两个表中都有元素未比完\*/

{

if (La.GetData(i) <= Lb.GetData(j))

{

Insert(La.GetData(i++), size);

}

else

{

Insert(Lb.GetData(j++), size);

}

}

while (i < La.Size())/\*Lb表的元素已经比完\*/

{

Insert(La.GetData(i++), size);

}

while (j < Lb.Size())/\*La表的元素已经比完\*/

{

Insert(Lb.GetData(j++), size);

}

}

template<typename DataType>void SeqList<DataType>::ChangeSize(int si)//修改顺序表的容量

{

if (si < size)cout << "SeqList<DataType>::ChangeSize出错！" << endl;

DataType \*temp;

temp = new DataType[si];

for (int i = 0; i < size; i++)//复制元素

{

temp[i] = list[i];

}

maxSize = si;

if(list!=NULL)delete []list;//删除原有数据区

list = temp;//指向新的数据区

}

template<typename DataType>SeqList<DataType>::SeqList(const SeqList<DataType>&other)

{

maxSize = other.maxSize;

size = other.size;

list = new DataType[maxSize];

for (int i = 0; i < size; i++)

{

list[i] = other.list[i];

}

}

template<typename DataType>

SeqList<DataType> & SeqList<DataType>::operator=(const SeqList<DataType> & other)

{

if (this == &other)return \*this;

maxSize = other.maxSize;

size = other.size;

delete[]list;

list = new DataType[maxSize];

for (int i = 0; i < size; i++)

{

list[i] = other.list[i];

}

return \*this;

}

template<typename DataType>

int SeqList<DataType>::GetIndex(const DataType & data)

{

for (int i = 0; i < size; i++)

{

if (list[i] == data)return i;

}

return -1;

}

//Question.h

#pragma once

#include<iostream>

#include<string>

using namespace std;

class Question/\*题目数据类型抽象\*/

{

public:

Question();

Question(string type, string question, string answer, int diffi);

Question(const Question & other);

Question& operator=(const Question &other);

~Question();

string GetType()const;//获取试题的类型

string GetQuestion()const;//获取试题的描述

string GetAnswer()const;//获取试题答案

int GetDifficulty()const;//获取试题难度系数

void Set(string, string, string, int);//可以修改试题的参数

bool operator<=(const Question &other);//重载的比较操作符，以问题难度m\_Difficulty为对象

bool operator<(const Question &other);

bool operator>=(const Question &other);

bool operator>(const Question &other);

bool operator==(const Question &other);

friend ostream &operator<<(ostream & out, const Question &que);//重载输出符号

private:

string m\_Type;//问题题型type，选择、填空、问答。。

string m\_Question;//问题描述question，

string m\_Answer;//答案answer，

int m\_Difficulty;//问题难度系数difficulty

};

//对试卷问题类型的输出格式的处理

void PrintQuestion(string str,string type);

//Question.cpp

#include "Question.h"

Question::Question()

{

m\_Type = " ";

m\_Question = " ";

m\_Answer = " ";

m\_Difficulty = 0;

}

Question::Question(string type, string question, string answer, int diffi)

{

m\_Type = type;

m\_Question = question;

m\_Answer = answer;

m\_Difficulty = diffi;

}

Question::Question(const Question & other)

{

m\_Type = other.m\_Type;

m\_Question = other.m\_Question;

m\_Answer = other.m\_Answer;

m\_Difficulty = other.m\_Difficulty;

}

Question& Question::operator = (const Question &other)

{

if (this == &other)return \*this;

m\_Type = other.m\_Type;

m\_Question = other.m\_Question;

m\_Answer = other.m\_Answer;

m\_Difficulty = other.m\_Difficulty;

return \*this;

}

Question::~Question()

{

}

string Question::GetType()const

{

return m\_Type;

}

string Question::GetQuestion()const//获取试题的描述

{

return m\_Question;

}

string Question::GetAnswer()const//获取试题答案

{

return m\_Answer;

}

int Question::GetDifficulty()const//获取试题难度系数

{

return m\_Difficulty;

}

void Question::Set(string type, string question, string answer, int diffi)//可以修改试题的参数

{

m\_Type = type;

m\_Question = question;

m\_Answer = answer;

m\_Difficulty = diffi;

}

bool Question::operator<=(const Question &other)//重载的比较操作符

{

if (m\_Difficulty <= other.m\_Difficulty)return true;

else return false;

}

bool Question::operator<(const Question &other)

{

if (m\_Difficulty < other.m\_Difficulty)return true;

else return false;

}

bool Question::operator>=(const Question &other)

{

if (m\_Difficulty >= other.m\_Difficulty)return true;

else return false;

}

bool Question::operator>(const Question &other)

{

if (m\_Difficulty > other.m\_Difficulty)return true;

else return false;

}

bool Question::operator==(const Question &other)

{

if (m\_Difficulty ==other.m\_Difficulty&&m\_Answer==other.m\_Answer

&&m\_Question==other.m\_Question&&m\_Type==other.m\_Type)return true;

else return false;

}

ostream &operator<<(ostream & out, const Question &que)//重载输出符号

{

out << "题目类型:" << que.m\_Type << "\n难度:" <<que.m\_Difficulty << " \n题目:";

PrintQuestion(que.m\_Question, que.m\_Type);

out<< " \n答案：" << que.m\_Answer;

return out;

}

//对试卷问题类型的输出格式的处理

void PrintQuestion(string str,string type)

{

if (type == "选择题")

{

size\_t index = 0;

for (size\_t i = 0; i < str.size()-1; i++)

{

if (str[i] == 'A'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

cout << temp << endl << "\t\t";

continue;

}

if (str[i] == 'B'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

cout << temp <<"\t";

continue;

}

if (str[i] == 'C'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

cout << temp << "\t";

continue;

}

if (str[i] == 'D'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

cout << temp <<"\t";

break;

}

}

string temp(str, index, str.size());

cout << temp << endl ;

}

else if (type == "填空题")cout << str << endl;

else cout << str << endl;

cout << endl;

}

//BasicLib.h

/\*基本题型库:

存放同一类试题

并从试题库中根据条件选取试题

\*/

#pragma once

#include"Question.h"

#include"Seqlist.h"

const int MAXSIZE = 100;

class BasicLib/\*基本题型库\*/

{

public:

BasicLib();

BasicLib(string );

BasicLib(const BasicLib & other);

~BasicLib();

//获取类型为str的number道题，str 为选择题，填空题，等，diffi2>=难度>=diffi1之间的题目，插尾放入list中

void GetQuestion(SeqList<Question>& list, string type, int diffi1, int diffi2, int number);

//获取类型为str的number道题，str 为选择题，填空题，等，diffi2>=难度>=diffi1之间的题目，插尾放入list中

void GetQuestion(SeqList<Question>& list, string type, int diffi1, int diffi2);

SeqList<Question> & GetSeqList();//获取顺序表的引用，用于顺序表的增加删除修改数据操作

string GetType()const;

void SetType(string str);

//提供关键词搜索功能

//获取类型为含有关键词key的所有类型为type的题，key为关键词，

//难度为插尾放入list中,如果type为空则搜索所有题型,

void GetQuestion(SeqList<Question>& list, int diffi1, int diffi2,string key,string type="" );

//输出基本库中所有数据

void print();

int GetQueTypeSize();//获取m\_QueType

void CalculateQueType();//计算m\_QueType

SeqList<string>m\_QueType;//记录问题类型总个数

private:

string m\_Type;//试题库类型，如语文，英语

SeqList<Question> m\_Que;//顺序表

};

//获取一个随机数，用于选题

double random(double start, double end);

int GetRandomNumber();

//BasicLib.cpp

#include "BasicLib.h"

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

BasicLib::BasicLib():m\_Que(MAXSIZE)

{

m\_Type = " ";

}

BasicLib::BasicLib(string str) : m\_Que(MAXSIZE)

{

m\_Type = str;

}

BasicLib::BasicLib(const BasicLib & other)

{

m\_Que = other.m\_Que;

m\_Type = other.m\_Type;

}

BasicLib::~BasicLib()

{

}

//获取类型为str的number道题，diffi2>=难度>=diffi1之间的题目，插尾放入list中

void BasicLib::GetQuestion(SeqList<Question>& list, string str, int diffi1, int diffi2, int number)

{

if (number > m\_Que.Size())cout << "BasicLib::GetQuestion error!" << endl;

int num = 0;

srand(unsigned(time(0)));

int array[MAXSIZE];//用于判断选出的题目是否重复

for (int i = 0; i < MAXSIZE; i++)

array[i] = -1;

for (int i = 0; i < number; )

{

num=(int) random(0, m\_Que.Size());//获取随机数

int diff = m\_Que.GetData(num).GetDifficulty();

string type = m\_Que.GetData(num).GetType();

//

if (num != array[num] &&type==str&& diff >= diffi1&&diff <= diffi2)

{

list.InsertEnd(m\_Que.GetData(num));

array[num] = num;

i++;//提取一个元素就加一

}

}

}

//获取所有类型为str题，diffi2>=难度>=diffi1之间的题目，插尾放入list中,str为空则获取所有类型的

void BasicLib::GetQuestion(SeqList<Question>& list, string str, int diffi1, int diffi2)

{

for (int i = 0; i <m\_Que.Size();i++)

{

int diff = m\_Que.GetData(i).GetDifficulty();

//获取所有类型为str题，diffi2>=难度>=diffi1之间的题目

if ( str.size()>0&& diff >= diffi1&&diff <= diffi2)

{

string type = m\_Que.GetData(i).GetType();

if (type==str)list.InsertEnd(m\_Que.GetData(i));

}

//str为空则获取所有类型的

else if( diff >= diffi1&&diff <= diffi2)

{

string type = m\_Que.GetData(i).GetType();

list.InsertEnd(m\_Que.GetData(i));

}

}

}

SeqList<Question> & BasicLib::GetSeqList()//获取顺序表的引用，用于顺序表的增加删除修改数据操作

{

return m\_Que;

}

string BasicLib::GetType()const

{

return m\_Type;

}

void BasicLib::SetType(string type)

{

m\_Type = type;

}

//在问题和答案中

//获取类型为含有关键词key的所有类型为type的题，key为关键词，难度为插尾放入list中,如果type为空则搜索所有题型

void BasicLib::GetQuestion(SeqList<Question>& list, int diffi1, int diffi2, string key, string type )

{

if (type.size() > 0)

{

for (int i = 0; i < m\_Que.Size(); i++)

{

Question temp = m\_Que.GetData(i);//取得第i个数据

string str = temp.GetType();

int diffi = temp.GetDifficulty();

if (str == type&&diffi >= diffi1&&diffi <= diffi2)

{

unsigned int loc1 = temp.GetAnswer().find(key, 0);//在答案中查找

unsigned int loc2 = temp.GetQuestion().find(key, 0);//在问题中查找

if (loc1 != string::npos||loc2 !=string::npos)//说明查找成功

list.InsertEnd(temp);//插入队列

}

}

}

else//否则查找特定的type题型类型

{

for (int i = 0; i < m\_Que.Size(); i++)

{

Question temp = m\_Que.GetData(i);//取得第i个数据

int diffi = temp.GetDifficulty();

if (diffi >= diffi1&&diffi <= diffi2)

{

unsigned int loc1 = temp.GetAnswer().find(key, 0);//在答案中查找

unsigned int loc2 = temp.GetQuestion().find(key, 0);//在问题中查找

unsigned int loc3 = temp.GetType().find(key, 0);//在类型中查找

if (loc1 != string::npos || loc2 != string::npos

||loc3!=string::npos)//说明查找成功

list.InsertEnd(temp);//插入队列

}

}

}

}

void BasicLib::print()

{

for (int i = 0; i < m\_Que.Size(); i++)

{

cout << m\_Que.GetData(i) << endl << endl;

}

}

int BasicLib::GetQueTypeSize()//获取m\_QueType

{

return m\_QueType.Size();

}

void BasicLib::CalculateQueType()//计算m\_QueType

{

int index = 0;

SeqList<int>arr;

for (int i = 0; i < m\_Que.Size(); i++)

{

if (i == 0)arr.InsertEnd(0);

string temp = m\_Que.GetData(i).GetType();

bool sym = true;

for (int j = 0; j < arr.Size(); j++)

{

if (temp == m\_Que.GetData(j).GetType())

{

sym = false;

}

}

if (sym == true)m\_QueType.InsertEnd(temp);

}

}

///随机数产生

double random(double start, double end)

{

return start + (end - start)\*rand() / (RAND\_MAX + 1.0);

}

int GetRandomNumber()

{

srand(unsigned(time(0)));

return int(random(0, MAXSIZE));

}

//Paper.h

#pragma once

#include"Paper.h"

#include"BasicLib.h"

class Paper

{

public:

Paper();

~Paper();

Paper(Paper&other);

string m\_Type;//试卷类型

string m\_Name;//试卷名字

SeqList<string>m\_QuestionType;//试卷试题类型容器

SeqList<int>m\_QuestionSize;//每一种试题的数目

SeqList<Question>m\_Question;//试卷试题容器

void Print(string type, string Qtype);//根据试卷类型type和试题类型Qtype打印试卷信息

void Print(bool answer);

string WriteFile(string type, string Qtype);//根据试卷类型type和试题类型Qtype将试卷信息写入文件

void WriteFile(bool answer);

//为试卷添加试题，

void AddQuestion(BasicLib&arr, string Qtype, int diffi1, int diffi2);

};

//Paper.cpp

#include "Paper.h"

#include<fstream>

#include<iostream>

using namespace std;

Paper::Paper() :m\_QuestionType(10), m\_Question(50)

{

m\_Name = "";

m\_Type = "";

}

Paper::~Paper()

{

WriteFile(true);

}

Paper::Paper( Paper &other)

{

m\_Name = other.m\_Name;

m\_Type = other.m\_Type;

for (int i = 0; i < other.m\_QuestionType.Size(); i++)

{

m\_QuestionType.InsertEnd(other.m\_QuestionType.GetData(i));

}

}

//试卷输出控制信息，第一个参数为试卷类型，第二个参数为试题类型('#'则输出试卷开始信息)

void Paper::Print(string type, string str)

{

if (type == "Chinese")

{

if (str == "#")cout << "\t\t\t\t语文考试卷\n \t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本语文试卷全卷"<<m\_QuestionType.Size()<<"种题型,"<<m\_Question.Size()

<<"小题。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else if (str == "解答题")cout << "第三题" << str << endl;

else cout << "第四题" << str << endl;

}

else if (type == "English")

{

if (str == "#")cout << "\t\t\t\t英语考试卷\n \t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本英语试卷全卷" << m\_QuestionType.Size() << "种题型" << m\_Question.Size()

<< "小题。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

else if (type == "Chinese")

{

if (str == "#")cout << "\t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本数学试卷全卷" << m\_QuestionType.Size() << "种题型" << m\_Question.Size()

<< "小题。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

else

{

if (type == "Chinese")

{

if (str == "#")cout << "\t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本"<<m\_Type<<"试卷全卷" << m\_QuestionType.Size() << "种题型" << m\_Question.Size()

<< "小题。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

}

}

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void Paper::Print( bool withAnswer)

{

int index = 0;

Print(m\_Type, "#");

for (int j = 0; j < m\_QuestionType.Size(); j++)

{

if (!withAnswer)Print(m\_Type,m\_QuestionType.GetData(j));

for (int i = 0; i<m\_Question.Size(); i++)

{

Question que = m\_Question.GetData(i);

if (withAnswer&&que.GetType() == m\_QuestionType.GetData(j))

cout << que << endl;

else PrintQuestion(que.GetQuestion(), que.GetType());

}

}

}

void Paper::AddQuestion(BasicLib&arr, string Qtype, int diffi1, int diffi2)

{

arr.GetQuestion(m\_Question, Qtype, diffi1, diffi2);

}

string Paper::WriteFile(string type, string str)

{

string str1="";

if (str == "#")

{

if (type == "Chinese")

{

str1 += "\t\t\t\t语文考试卷\n";

}

else if (type == "English")

{

str1 += "\t\t\t\t英语考试卷\n";

}

else if (type == "Math")

{

str1 += "\t\t\t\t数学考试卷\n";

}

str1 += "\n \t\t 说明：1.答题前，请将姓名、考生号、";

str1 += "\n 考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。";

str1 += "\t\t2 本语文试卷考试时间100分钟，满分100分。";

str1 += "\n \t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。";

str1 += "\n \t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑";

str1 += "如需改动，用橡皮擦干净后，再涂选其它答案；";

str1 += "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。";

str1 += "\n\t\t 5.考试结束，请将本试卷和答题卡一并交回。\n\n";

}

else if (str == "选择题")str1+="第一题" + str +":请根据以下选项选出最佳答案\n";

else if (str == "填空题")str1+= "第二题"+str + ":请在以下空格处填写最佳答案\n" ;

else if (str == "解答题")str1+= "第三题" +str +"\n";

else cout << "第四题" << str << endl;

return str1;

}

string PrintQue(string str, string type)

{

string str1;

if (type == "选择题")

{

size\_t index = 0;

for (size\_t i = 0; i < str.size() - 1; i++)

{

if (str[i] == 'A'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

str1+= temp +"\n \t\t";

continue;

}

if (str[i] == 'B'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

str1 += temp+"\t";

continue;

}

if (str[i] == 'C'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

str1 += temp +"\t";

continue;

}

if (str[i] == 'D'&&str[i + 1] == '.')

{

string temp(str, index, i - index);

index = i + 2;

str1 += temp +"\t";

break;

}

}

string temp(str, index, str.size());

str1 += temp +"\n";

}

else if (type == "填空题")str1+= str +"\n";

else str1 += str +"\n";

return str1 += "\n";

}

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void Paper::WriteFile(bool withAnswer)

{

int index = 0;

fstream ifile;

ifile.open(m\_Name + ".txt");

if (!ifile)throw("Paper::WriteFile error!");

string str = WriteFile(m\_Type, "#");

for (int j = 0; j < m\_QuestionType.Size(); j++)

{

if (!withAnswer)Print(m\_Type, m\_QuestionType.GetData(j));

for (int i = 0; i<m\_Question.Size(); i++)

{

Question que = m\_Question.GetData(i);

if (withAnswer&&que.GetType() == m\_QuestionType.GetData(j))

{

ifile << que << endl;

}

else ifile << PrintQue(que.GetQuestion(), que.GetType()) << endl;

}

}

}

//TestLib.h

/\*试题总库类

包含多种基本试题库放在m\_Basiclib 顺序表中\*/

/\*每一种试题类型都使用英文索引，如Chinese，English等，对应的中文索引在MFC中转变\*/

/\*为节省空间，m\_Basiclib 顺序表中只有三个基本库

如果要对其他库操作就重新读取文件，循环存于m\_Basiclib 顺序表中

\*/

#pragma once

#include"Seqlist.h"

#include"BasicLib.h"

#include"Paper.h"

class TestLib //\*试题总库类

{

public:

TestLib();//初始化时只设置3种试题类型,English,Chinese,Math

~TestLib();

//添加试题que，type为试题属于的大类，如语文

void AddQuestion(string type, Question &que);

//删除试题，type为大类，index为其在基本库中的索引

void DeleteQuestion(string type, int index);

//获取总库中index位置的试题大类信息

string GetTpye(int index);

//获取总库中类型为type的基本类的位置，找不到就返回-1；

int GetIndex(string type);

////添加基本试题库que，type为试题属于的大类，如语文

void AddBasicLib(string type);

//删除基本试题库试题，type为大类

void DeleteBasicLib(string type);

//根据试题类型type读写文件操作,type 为试题大类，如语文

int ReadFile(string type);//读取文件，并返回其在m\_Basiclib列表中的位置

void WriteFile(string type);

void DeleteFile(string type);//删除名为type的txt文件

bool AlreadyHave(string type);//判断是否已经存在类型为type的基本试题类

void SetNextIndex();

void Print(string type);//输出type类型的基本试题库

//SeqList<BasicLib>&GetBasicLib(int index);//获取m\_BasicLiib库中的第index个数据

SeqList<BasicLib> m\_Basiclib;//基本库列表，初始化只包含3个元素，

SeqList<string> m\_Type;//基本试题库的所有已有类型

int m\_NextIndex;//标志m\_BasicLib的下标，用于3个位置的循环放置，值为0，1，2,使用一次就要改变一次值

//生成试卷模块

public:

SeqList<BasicLib> m\_Testpapers;//试卷容器

//在试卷中添加题目

void AddTestPaperQuesion(string papertype, string questionType, int diffi1, int diffi2,int number);

//在容器中添加试卷

void AddTestPaper(string PaperType,string PaperName);

SeqList<string> m\_PaperName;//试卷名字列表

//获取试卷下标

int GetPaperIndex(string PaperName);

//读写试卷信息

void ReadPaper(string PaperName);

void WritePaper(string PaperType, string PaperName);

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void PrintPaper(string PaperType, string PaperName,bool withAnswer);

//生成试卷模块

public:

SeqList<Paper>m\_MyPaper;

//在试卷中添加题目

void AddTestPaperQuesion(string papertype, string questionType, int diffi1,

int diffi2, int number,string str);

//在容器中添加试卷

void AddTestPaper(string PaperType, string PaperName, string str );

//获取试卷下标

int GetPaperIndex(string PaperName, string str);

//读写试卷信息

void ReadPaper(string PaperName, string str);

void WritePaper(string PaperType, string PaperName, string str );

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void PrintPaper(string PaperType, string PaperName, bool withAnswer, string str);

};

//TestLib.cpp

#include "TestLib.h"

#include<iostream>

#include<fstream>

using namespace std;

//#define MYDEBUG

TestLib::TestLib():m\_Basiclib(3),m\_Type(3),m\_Testpapers(10),m\_PaperName(10)//初始化时只设置3种试题类型

{

m\_NextIndex = 0;

fstream ifile;

ifile.open("type.txt");//读取基本库

if (!ifile )throw("TestLib::ReadPaper error!");

string type;

while (true)

{

if (ifile.eof())break;

getline(ifile, type);

m\_Type.InsertEnd(type);

}

ifile.close();

ifile.open("papername.txt");//读取试卷库

if (!ifile)throw("TestLib::ReadPaper error!");

while (true)

{

if (ifile.eof())break;

getline(ifile, type);

m\_PaperName.InsertEnd(type);

}

ifile.close();

m\_Basiclib.InsertEnd(\*new BasicLib("English"));

m\_Basiclib.InsertEnd(\*new BasicLib("Chinese"));

m\_Basiclib.InsertEnd(\*new BasicLib("Math"));

ReadFile("English");

ReadFile("Chinese");

ReadFile("Math");

}

TestLib::~TestLib()

{

fstream ofile;

ofile.open("type.txt");

string type;

for (int i = 0; i < m\_Type.Size();i++)

{

ofile << type << endl;

}

ofile.close();

for (int i=0; i < m\_Basiclib.Size(); i++)

{

WriteFile(m\_Basiclib.GetData(i).GetType());//将m\_BasicLib表的数据写入文件中保存

}

//保存试卷库

ofile.open("papername.txt");

for (int i = 0; i < m\_Type.Size(); i++)

{

ofile << type << endl;

}

ofile.close();

for (int i = 0; i < m\_Testpapers.Size(); i++)

{

//将m\_Testpapers表的数据写入文件中保存

WritePaper(m\_Testpapers.GetData(i).GetType(),m\_Type.GetData(i));

}

}

//添加试题que，type为试题属于的大类，如Chinese,type必须已经存在的

void TestLib::AddQuestion(string type, Question &que)

{

int i = GetIndex(type);

if (i!=-1)m\_Basiclib.GetData(i).GetSeqList().InsertEnd(que);//插入基本试题库的尾部

else cout << " TestLib::AddQuestion 插入试题失败!" << endl;

}

//删除试题，type为大类，index为其在基本库中的索引

void TestLib::DeleteQuestion(string type, int index)

{

int i = GetIndex(type);

if (i != -1)m\_Basiclib.GetData(i).GetSeqList().Delete(index);//删除

}

//获取总库中index位置的试题大类信息

string TestLib::GetTpye(int index)

{

return m\_Basiclib.GetData(index).GetType();

}

//获取总库中类型为type的基本类的位置,找不到就返回-1；

int TestLib::GetIndex(string type)

{

if (AlreadyHave(type))//如果总库中存在type类型的试题库

{

bool sym = false;

int i = 0;

for (; i < m\_Basiclib.Size(); i++)

{

if (m\_Basiclib.GetData(i).GetType() == type)//找到试题大类

{

sym = true; break;

}

}

if(sym==true)return i;//说明在m\_Basiclib列表中

else return ReadFile(type);//说明不在m\_Basiclib列表中，需要从文件中重新读取

}

else return -1;

}

////添加基本试题库que，type为试题属于的大类，如Chinese ，type为不存在的

void TestLib::AddBasicLib(string type)

{

WriteFile(m\_Basiclib.GetData(m\_NextIndex).GetType());//现将m\_NextIndex位置的内容写入txt文件中保存

m\_Basiclib.GetData(m\_NextIndex).GetSeqList().MoreDelete

(0, m\_Basiclib.GetData(m\_NextIndex).GetSeqList().Size()-1);//清除m\_NextIndex位置原有内容

m\_Basiclib.GetData(m\_NextIndex).SetType(type);//在m\_NextIndex位置建立type试题类

m\_Type.InsertEnd(type);

SetNextIndex();

}

//删除基本试题库试题，type为大类

void TestLib::DeleteBasicLib(string type)

{

if (AlreadyHave(type))

{

int i = GetIndex(type);

m\_Basiclib.GetData(i).GetSeqList().MoreDelete

(0, m\_Basiclib.GetData(i).GetSeqList().Size() - 1);//清除所有内容

DeleteFile(type);

for (int i = 0; i < m\_Type.Size(); i++)//删除类型总表中的

{

if (m\_Type.GetData(i) == type)

{

m\_Type.Delete(i);

}

}

m\_NextIndex = i;//m\_NextIndex指向空闲的位置

}

}

//根据试题类型type读写文件操作,type 为试题大类，如语文,返回其存于m\_BasicLib中的位置

int TestLib::ReadFile(string type)

{

fstream ifile;

ifile.open(type+".txt");

if (!ifile )throw("TestLib::ReadPaper error!");

int index = m\_NextIndex;//获取当前存取位置

m\_Basiclib.GetData(index).SetType(type);//?

string ty, question, answer;

int difficulty;

while (true)

{

if (ifile.eof())break;

getline(ifile, ty);

getline(ifile, question);

getline(ifile, answer);

ifile >> difficulty;

ifile.get();

#ifdef MYDEBUG

cout << "type:" << ty << "\n question:" << question << "\n answer:" <<

answer << "\n difficulty:" << difficulty << endl;

#endif

Question que(ty, question, answer, difficulty);

m\_Basiclib.GetData(index).GetSeqList().InsertEnd(que);

}

//m\_Basiclib.GetData(index).print();

ifile.close();

SetNextIndex();//存储位置变化

return index;

}

void TestLib::WriteFile(string type)

{

fstream ofile;

ofile.open(type+".txt", ios::out);

if (!ofile)throw("TestLib::ReadPaper error!");

int index = GetIndex(type);//获取当前存取位置

if (index == -1)return;

for (int i = 0; i<m\_Basiclib.GetData(index).GetSeqList().Size();i++)

{

Question que = m\_Basiclib.GetData(index).GetSeqList().GetData(i);

ofile << que.GetType() << endl << que.GetQuestion() <<endl <<

que.GetAnswer() << endl << que.GetDifficulty() << endl;

}

ofile.close();

m\_NextIndex=index;//存储位置变化

}

void TestLib::DeleteFile(string type)//删除名为type的txt文件

{

const char \*ptr=type.c\_str();

//删除名为type的文件

if (!remove(ptr))cout << "TestLib::DeleteFile Error!" << endl;

//On success, zero is returned. On error, -1 is returned, and errno is set appropriately

}

bool TestLib::AlreadyHave(string type)//判断是否已经存在类型为type的基本试题类

{

for (int i = 0; i < m\_Type.Size(); i++)

{

if (m\_Type.GetData(i) == type)

{

return true;

}

}

return false;

}

void TestLib::SetNextIndex()

{//标志m\_BasicLib的下标，用于3个位置的循环放置，值为0，1，2,使用一次就要改变一次值

m\_NextIndex++;

m\_NextIndex = m\_NextIndex % 3;

}

void TestLib::Print(string type)

{

int index = GetIndex(type);

if(index!=-1)m\_Basiclib.GetData(index).print();

}

//获取m\_BasicLiib库中的第index个数据

/\*SeqList<BasicLib> &TestLib::GetBasicLib(int index)

{

return m\_Basiclib.GetData(index);

}\*/

//在试卷中添加题目

void TestLib::AddTestPaperQuesion(string paperType,string questionType,

int diffi1, int diffi2,int number)

{

int index = GetIndex(paperType);

int index2 = GetPaperIndex(paperType);

if (index == -1||index2==-1)throw("TestLib::AddTestPaperQuesion error!");

SeqList<Question>temp;

m\_Basiclib.GetData(index).GetQuestion(temp, questionType, diffi1, diffi2,number);

for (int i = 0; i < temp.Size(); i++)

{

m\_Testpapers.GetData(index2).GetSeqList().InsertEnd(temp.GetData(i));

}

}

//在容器中添加试卷

void TestLib::AddTestPaper(string PaperType, string PaperName)

{

int index = m\_Testpapers.Size();

m\_Testpapers.InsertEnd(\*new BasicLib(PaperType));

string temp = m\_PaperName.GetData(index);

m\_PaperName.SetItem(PaperName,index);

m\_PaperName.InsertEnd(temp);

}

//获取试卷在试卷名表中下标,获取失败返回-1

int TestLib::GetPaperIndex(string PaperName)

{

int i = 0;

bool sym = false;

for (; i < m\_PaperName.Size(); i++)

{

if (PaperName == m\_PaperName.GetData(i))

{

sym = true;

return i;

}

}

return -1;

}

//读写试卷信息

void TestLib::ReadPaper(string PaperName)

{

fstream ifile;

ifile.open(PaperName + ".txt");

if (!ifile||ifile.eof())throw("TestLib::ReadPaper error!");

int index = m\_Testpapers.Size();//获取插入位置

for (int i = 0; i < m\_PaperName.Size(); i++)//试卷名表中的顺序要和试卷容器中的顺序一致

{

if (PaperName == m\_PaperName.GetData(i))

{

string temp = m\_PaperName.GetData(i);

m\_PaperName.SetItem(PaperName, index);

m\_PaperName.SetItem(temp, i);

}

}

string papertype;

getline(ifile, papertype);//第一行存放的是试卷类型信息

m\_Testpapers.InsertEnd(\*new BasicLib(papertype));//插入试卷

string ty, question, answer;

int difficulty;

while (true)

{

if (ifile.eof())break;

getline(ifile, ty);

getline(ifile, question);

getline(ifile, answer);

ifile >> difficulty;

ifile.get();

#ifdef MYDEBUG

cout << "type:" << ty << "\n question:" << question << "\n answer:" <<

answer << "\n difficulty:" << difficulty << endl;

#endif

Question que(ty, question, answer, difficulty);

m\_Testpapers.GetData(index).GetSeqList().InsertEnd(que);

}

#ifdef MYDEBUG

m\_Testpapers.GetData(index).print();

#endif

ifile.close();

return ;

}

void TestLib::WritePaper(string PaperType, string PaperName)

{

fstream ofile;

ofile.open(PaperName + ".txt", ios::out);

if (!ofile)throw("TestLib::WritePaper error!");

ofile << PaperType << endl;

int index = GetPaperIndex(PaperName);//获取当前存取位置

if (index == -1 || index >=m\_Testpapers.Size())

{

ofile.close();

throw("TestLib::WritePaper error!");

}

for (int i = 0; i<m\_Testpapers.GetData(index).GetSeqList().Size(); i++)

{

Question que = m\_Testpapers.GetData(index).GetSeqList().GetData(i);

ofile << que.GetType() << endl << que.GetQuestion() << endl <<

que.GetAnswer() << endl << que.GetDifficulty() << endl;

}

ofile.close();

}

//试卷输出控制信息，第一个参数为试卷类型，第二个参数为试题类型('#'则输出试卷开始信息)

void print(string type,string str)

{

if (type == "Chinese")

{

if (str == "#")cout << "\t\t\t\t语文考试卷\n \t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本语文试卷全卷四大题，共4页。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else if(str=="解答题")cout << "第三题" << str << endl;

else cout << "第四题" << str << endl;

}

else if (type == "English")

{

if (str == "#")cout << "\t\t\t\t英语考试卷\n \t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本英语试卷全卷四大题，共4页。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

else if (type == "Chinese")

{

if (str == "#")cout << "\t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本语文试卷全卷四大题，共4页。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

else

{

if (type == "Chinese")

{

if (str == "#")cout << "\t\t 说明：1.答题前，请将姓名、考生号、考场、试室号和座位号用规定的笔写在答题卡指定的位置上，将条形码粘贴好。"

<< endl << "\t\t2 本语文试卷全卷四大题，共4页。考试时间100分钟，满分100分。"

<< endl << "\t\t 3.考生必须在答题卡上按规定作答；凡在试卷、草稿纸上作答的，其答案一律无效。答题卡必须保持清洁，不能折叠。"

<< endl << "\t\t 4.选择题每小题选出答案后，用2B铅笔将答题卡选择题答题区内对应题目的答案标号涂黑，"

<< "如需改动，用橡皮擦干净后，再涂选其它答案；"

<< "非选择题答案必须用规定的笔，按作答题目序号，写在答题卡非选择题答题区内，作文写在非选择题区的作文格内。"

<< endl << "\t\t 5.考试结束，请将本试卷和答题卡一并交回。" << endl << endl;

else if (str == "选择题")

cout << "第一题" << str << ":请根据以下选项选出最佳答案" << endl;

else if (str == "填空题")

cout << "第二题" << str << ":请在以下空格处填写最佳答案" << endl;

else cout << "第三题:" << str << endl;

}

}

}

//打印试卷，bool参数为true则连答案信息也打印，否则只打印问题；

void TestLib::PrintPaper(string PaperType, string PaperName, bool withAnswer)

{

int index = GetPaperIndex(PaperName);//获取当前存取位置

if (index == -1 || index >= m\_Testpapers.Size())

{

throw("TestLib::WritePaper error!");

}

int size = m\_Testpapers.GetData(index).GetSeqList().Size();

int index2 = 0;//记录问题类型

for (int j = 0; j < m\_Testpapers.GetData(index).m\_QueType.Size(); j++)

{

string str = m\_Testpapers.GetData(index).m\_QueType.GetData(j);

if (!withAnswer)cout << j<<"."<<str << endl;

for (int i = 0; i<size; i++)

{

Question que = m\_Testpapers.GetData(index).GetSeqList().GetData(i);

if (withAnswer&&que.GetType() == str)

cout << que << endl;

else PrintQuestion(que.GetQuestion(), que.GetType());

}

}

for (int i = 0; i<size; i++)

{

Question que = m\_Testpapers.GetData(index).GetSeqList().GetData(i);

if (withAnswer)cout << que;

else PrintQuestion(que.GetQuestion(), que.GetType());

}

}

//在试卷中添加题目

void TestLib::AddTestPaperQuesion(string papertype, string questionType, int diffi1,

int diffi2, int number, string papername)

{

int index = GetPaperIndex(papername,papertype);

int index2 = GetIndex(papertype);

if (index == -1||index2==-1)throw("TestLib::AddTestPaperQuesion");

m\_MyPaper.GetData(index).AddQuestion(m\_Basiclib.GetData(index2), questionType, diffi1, diffi2);

}

//在容器中添加试卷

void TestLib::AddTestPaper(string PaperType, string PaperName, string str)

{

Paper temp;

temp.m\_Name = PaperName;

temp.m\_Type = PaperType;

m\_MyPaper.InsertEnd(temp);

}

//获取试卷下标

int TestLib::GetPaperIndex(string PaperName, string str)

{

for (int i = 0; i < m\_MyPaper.Size(); i++)

{

if (PaperName == m\_MyPaper.GetData(i).m\_Name)return i;

}

return -1;

}

//Test.h

#pragma once

#include"BasicLib.h"

#include"Question.h"

#include"Seqlist.h"

class Test

{

public:

Test();

~Test();

void testLib();//管理员模块

void userLib();//普通用户模块

};

#include "Test.h"

#include"TestLib.h"

#include<iostream>

#include<vector>

#include<string>

using namespace std;

//#define MYDEBUG

Test::Test()

{

}

Test::~Test()

{

}

void Test::question()

{

}

void Test::bascilib()

{

BasicLib BLib;

BLib.SetType("math");

for (int i = 0; i < 20; i++)

{

Question ques;

ques.Set("填空题", "1+1=\_\_\_", "2", i);

BLib.GetSeqList().InsertEnd(ques);

}

BLib.print();

SeqList<Question> my;

BLib.GetQuestion(my, "填空题", 2, 5, 3);

my.Print();

cout << BLib.GetType() << endl;

BLib.GetQuestion(my, 10, 15, "1");

my.Print();

}

void Test::seqlist()

{

SeqList<int >arr(10);

for (int i = 0; i <10; i++)

{

arr.Insert(10 - i,i);

}

for (int i = 0; i < arr.Size(); i++)

{

cout<<arr.GetData(i)<<" ";

}cout << endl;

arr.Insert(11,8);

arr.InsertEnd(12);

for (int i = 0; i < arr.Size(); i++)

{

cout << arr.GetData(i) << " ";

}cout << endl<<arr.Size();

}

void Add(TestLib &my)

{

while (true)

{

cout << endl << "\t\t\t\t--------添加模块----------" << endl;

cout << "\t\t\t\t-----1.添加基本试题库-----" << endl;

cout << "\t\t\t\t-----2.添加试题--------------" << endl;

cout << "\t\t\t\t-----3.退出-------" << endl;

cout << "\t\t\t\t-----请选择功能：" ;

char number;

cin >> number;

switch (number)

{

case '1':

{

cout << "\t\t\t\t请输入试卷类型(英文)：";

string type;

cin >> type;

if (my.AlreadyHave(type))

{

cout << type << "\t\t\t\t类型的试题库已经存在！" << endl;

break;

}

my.AddBasicLib(type);

cout << "\t\t\t\t添加成功！" << endl;

}break;

case '2':

{

cout << "\t\t\t\t请输入试卷类型(英文)：";

string type;

cin >> type;

if (!my.AlreadyHave(type))

{

cout << type << "\t\t\t\t类型的试题库不存在，是否添加到基本库？(y/n)：" ;

char ptr;

cin >> ptr;

if (ptr == 'y')my.AddBasicLib(type);

else break;

}

int index = my.GetIndex(type);

string ty, quesion, answer;

int difficulty;

cout << "\t\t\t\t请输入试题类型：";

cin >> ty;

cout << "\t\t\t\t请输入试题题目：";

getchar();

getline(cin,quesion);

cout << "\t\t\t\t请输入试题答案：";

getline(cin,answer);

cout << "\t\t\t\t请输入试题难度：";

cin >> difficulty;

Question temp(ty, quesion, answer, difficulty);

my.AddQuestion(type, temp);

cout << "\t\t\t\t添加成功！" << endl;

}break;

case '3':break;

default:cout << "\t\tv输入错误！" << endl;

break;

}

char ptr;

cout << "\t\t\t\t----是否继续添加？（y/n）：";

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

}

void Delete(TestLib &my)

{

while (true)

{

bool sym = true;

cout << endl<<"\t\t\t\t--------删除模块------------" << endl;

cout << "\t\t\t\t-----1.添加基本试题库或试题-------" << endl;

cout << "\t\t\t\t-----2.删除基本试题库或试题-------" << endl;

cout << "\t\t\t\t-----3.生成试卷-------------------" << endl;

cout << "\t\t\t\t-----4.查看数据库-----------------" << endl;

cout << "\t\t\t\t-----5.退出系统-------------------" << endl;

cout << "\t\tv\t\t--------请选择功能：--------------" << endl;

char number;

cin >> number;

switch (number)

{

case '1':break;

case '2':break;

case '3':break;

case '4':break;

case '5':sym = false; break;

default:cout << "输入错误！" << endl;

break;

}

}

}

void Create(TestLib &my)

{

while (true)

{

string name;

cout << "\t\t\t\t--------生成试卷模块----------" << endl;

if (my.m\_Type.Size() > 0)cout << "当前试题库中共有" << my.m\_Type.Size() << "种大类试题："

<< "\n分别为：";

for (int i = 0; i < my.m\_Type.Size(); i++)

{

cout << my.m\_Type.GetData(i) << ",";

}

cout << endl << "\t\t\t\t-----请输入你要生成的试卷类型：";

string type;

cin >> type;

int index = my.GetIndex(type);

if (index == -1)

{

cout << "\t\t\t\t输入错误！请重新输入！" << endl;

continue;

}

cout << "\t\t\t\t-----请输入试卷名称：" ;

cin >> name;

my.AddTestPaper(type, name);

cout << "-\t\t\t\t---你想要生成" << type << "类型的试卷" << endl;

while (true)

{

string ty;

int diffi1, diffi2,number;

cout << "-\t\t\t\t-------请为试卷添加试题-------" << endl;

cout << "\t\t\t\t-----请输入将要添加的试题类型：";

cin >> ty;

cout << "-\t\t\t\t----请输入该试题数目：";

cin >> number;

cout << "\t\t\t\t-----请输入试题难度范围-------" << endl;

cin >> diffi1 >> diffi2;

my.AddTestPaperQuesion(type, name, diffi1, diffi2, number, name);

cout << "\t\t\t\t是否继续为该试卷添加试题?(y/n):";

char ptr;

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

cout << "\t\t\t\t是否继续添加试卷?(y/n):";

char ptr;

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

}

void Scan(TestLib &my){

while (true)

{

cout << "\t\t\t\t--------查看模块----------" << endl;

if (my.m\_Type.Size() > 0)cout << "\t\t\t\t当前试题库中共有" << my.m\_Type.Size() << "种大类试题："

<< "\n分别为：";

for (int i = 0; i < my.m\_Type.Size(); i++)

{

cout << my.m\_Type.GetData(i)<<",";

}

cout <<endl<< "\t\t\t\t-----请输入你要查看的大类：";

string type;

cin >> type;

int index = my.GetIndex(type);

if (index==-1)

{

cout << "输入错误！请重新输入！" << endl;

continue;

}

cout << "\t\t\t\t你想要查看" << type << "类型的基本库" << endl;

while (true)

{

cout << "\t\t\t\t---------1.查看所有库内容--------" << endl;

cout << "\t\t\t\t---------2.按条件查看------------" << endl;

cout << "\t\t\t\t---------3.按关键词查找----------" << endl;

cout << "\t\t\t\t---------4.退出------------------" << endl;

char sym;

cin >> sym;

switch (sym)

{

case'1':

cout <<"\t\t\t----------生成的试卷如下：------------"<< endl;

my.m\_Basiclib.GetData(index).print();

break;

case'2':

{

cout << "\t\t---请输入查看条件：" << endl;

cout << "\t\t----请输入查看试题类型(输入'#'为查看所有类型):";

string type;

cin >> type;

if (type == "#")type = "";

int diffi1, diffi2;

cout << "\t\t---请输入试题难度（两个非负整数）：" ;

cin >> diffi1 >> diffi2;

SeqList<Question> temp(100);

my.m\_Basiclib.GetData(index).GetQuestion(temp,type,diffi1,diffi2);

temp.Print();

}break;

case'3':

{

cout << "\t\t------请输入关键字:";

string key;

cin >> key;

SeqList<Question> temp(100);

my.m\_Basiclib.GetData(index).GetQuestion(temp, 0, 10,key);

temp.Print();

}break;

case'4':break;

default:cout << "输入有误！" << endl;

break;

}

char ptr;

cout << "\t\t----继续查看"<<type<<"类型？（y/n）：";

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

char ptr;

cout << "\t\t----是否继续查看？（y/n）：";

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

}

void ScanPaper(TestLib&my)

{

while (true)

{

cout << "\t\t\t\t--------查看试卷模块----------" << endl;

if (my.m\_MyPaper.Size() > 0)cout << "当前试题库中共有" << my.m\_Type.Size() << "套试卷"

<< "\n分别为：";

else

{

cout << "\t\t当前没有试卷！请等待管理员添加！" << endl;

break;

}

for (int i = 0; i < my.m\_MyPaper.Size(); i++)

{

cout << i+1<<"："<<my.m\_MyPaper.GetData(i).m\_Name << endl;

}

cout << endl << "\t\t\t\t-----请选择你要查看的试卷：";

int number;

cin >> number;

my.m\_MyPaper.GetData(number-1).Print(false);

char ptr;

cout << "\t\t\t\t----是否继续查看？（y/n）：";

cin >> ptr;

if (ptr == 'y')continue;

else break;

}

}

void Test::testLib()

{

/\*试卷格式设置、试题范围设置等；

生成试卷和生成历史试卷模块包括接收相应的参数、

打开题库、生成相应试卷、把试卷的全部信息存档；

试卷输出模块包括按要求输出相应试卷；

\*/

TestLib my;

#ifdef MYDEBUG

cout << endl<<"Test::testLib()" << endl << endl;

//my.Print("Math");

#endif

cout << "--------欢迎管理员！----------" << endl;

while (true)

{

bool sym = true;

cout << "\t\t\t\t-----1.添加基本试题库或试题-------" << endl;

cout << "\t\t\t\t-----2.删除基本试题库或试题-------" << endl;

cout << "\t\t\t\t-----3.生成试卷-------------------" << endl;

cout << "\t\t\t\t-----4.查看试题库-----------------" << endl;

cout << "\t\t\t\t-----5.查看所有试卷-------------------" << endl;

cout << "\t\t\t\t-----6.退出系统-------------------" << endl;

cout << "\t\t\t\t--------请选择功能：--------------" << endl;

char number;

try{

cin >> number;

switch (number)

{

case '1':Add(my); break;

case '2':Delete(my); break;

case '3':Create(my); break;

case '4':Scan(my); break;

case '5':ScanPaper(my); break;

case '6':sym = false; break;

default:cout << "\t\t\t\t输入错误！" << endl;

break;

}

}

catch (...)

{

cout << "\t\t\t\t发生错误！" << endl;

}

if (sym == true)

{

cout << "\t\t\t\t----是否回到主页？（y/n）：";

cin >> number;

}

if (number == 'y')continue;

else break;

}

}

void Test::userLib()

{

TestLib my;

#ifdef MYDEBUG

cout << endl << "Test::testLib()" << endl << endl;

//my.Print("Math");

#endif

cout << "--------欢迎普通用户登录！----------" << endl;

while (true)

{

bool sym = true;

cout << "-----1.生成试卷-------------------" << endl;

cout << "-----2.查看试题库-----------------" << endl;

cout << "-----3.查看库中的试卷-------------" << endl;

cout << "-----4.退出系统-------------------" << endl;

cout << "--------请选择功能：--------------" << endl;

char number;

try

{

cin >> number;

switch (number)

{

case '1':Create(my); break;

case '2':Scan(my); break;

case '3':ScanPaper(my); break;

case '4':sym = false; break;

default:cout << "输入错误！" << endl;

break;

}

}

catch (...)

{

cout << "-----发生错误！正在恢复..." << endl;

cout << "------恢复完成！";

}

if (sym == true)

{

cout << "----是否回到主页？（y/n）：";

cin >> number;

}

if (number == 'y')continue;

else break;

}

}

**参考资料：**

**数据结构（c++语言描述）朱战立；**

**C++语言程序设计**

**C++ primer 第六版**