

题目：学生健康管理系统--二叉树

学生姓名：郑嘉涛

学号： 20152100015

专业班级：计算机类5班

指导教师：陈红英

学生健康系统--二叉树

摘 要

目前人们对于健康的关注程度越来越高，而同时学校对于学生的健康也变得更加关注，但是对学生的健康档案管理却没有一个较好的解决办法，基于此，我们完成了用二叉树实现的学生健康管理系统。二叉树是一种非线性的数据结构，使用二叉树来完成有多种优势，二叉树是动态的，可以随着学生的数量改变而改变，更好的使用空间，防止空间浪费；二叉树具有多种优点，可以保存许多信息，而且二叉树的多种遍历方式可以满足对数据不同的查看需要，二叉树也可以将学生进行归类管理，更方便了我们访问不同的学生。而且完全二叉树转换成顺序结构也方便，而且操作上也更简单，但是考虑到多种原因，此处不采用完全二叉树，而仅仅使用了简单二叉树，且在二叉树的创建过程中需要注意，二叉树的创建方式多样，比如使用带空节点的前序遍历创建二叉树、用前序遍历和中序遍历或者用中序遍历和后序遍历创建二叉树，因此在开发时需特别注意选择的创建方法。而在文件的写入方面，写入后创建的二叉树和原先的二叉树不一定完全相同，但是使用上区别不大。在开发的过程中使用了C++的面向对象特性，简化了开发过程。

**关键词**：C++；面向对象；二叉树；

前言

**实验目的：**

开发一个可供日常使用的学生健康管理系统，使用二叉树进行开发。开发的学生健康管理系统不仅能够满足日常需要，而且还能方便的扩展为其他系统，但是需要进行一些改动。在开发的过程中，通过对二叉树的不断操作，学习二叉树的使用，加深对二叉树的理解等。

**实验意义：**

在实验的过程中，通过创建二叉树并完成基本方法，以及对二叉树的不断操作，能够学习二叉树的使用，加深对二叉树的理解。并且开发一个可供日常使用的学生健康管理系统，该系统能够方便的扩展为其他系统，其中开发的二叉树也能作为其他的基础。

**实验内容概述：**

实验先建立了一个学生类，作为实验的基本数据类型，学生类中含有多种数据，包括学生的学号，姓名，出生日期，性别，健康状况等信息。

学生类的基本及主要内容：

#ifndef \_STUDENT\_H

#define \_STUDENT\_H

#include<iostream>

#include<fstream>

using namespace std;

class Student{

private:

char Id[13];

char name[16];

char birth[10];

int sex;

int health;

public:

//构造函数

Student(char \*Id = NULL , char \*name = NULL , char \*birth = NULL , int sex = 0 , int health = 0);

Student(Student &t);

~Student();

//输出student对象

void display();

bool writeToFile(char \*address = "C:\\student.txt");

bool readFromFile(char \*address = "C:\\student.txt");

friend Student\* newStudent();

};

#endif

其中的newStudent()，是用来创建学生的对象的函数，里面有一些提示，函数会返回一个学生对象的指针。函数如下：

Student\* newStudent(){

char id[13];

char name[16];

char birth[10];

int sex ;

int health;

cin.sync();

cout<<"请输入学生的学号：";

cin>>id;

while(strlen(id) != 11){

cout<<"重新输入学号：";cin>>id;

}

if(strcmp(id , "00000000000") == 0){

Student \*student = new Student(id , "xx" , "12345678" , 1 , 1);

return student;

}

cout<<"请输入学生的姓名：";

cin.sync();

cin>>name;

cout<<"请输入学生的出生日期：";

cin>>birth;

while(strlen(birth) != 8){

cout<<"重新输入出生日期：";cin>>birth;

}

cout<<"请输入学生的性别：";

cin.sync();

cin>>sex;

while(sex != 1 && sex != 2){

cout<<"请重新输入学生的性别：";cin>>sex;

}

cout<<"请输入学生的健康状况：";

cin.sync();

cin>>health;

Student \*student = new Student(id , name , birth , sex , health);

return student;

}

二叉树节点，使用了结构体来构建节点，使得节点可以直接访问，操作上方便，节点包含基本的数据，包括左孩子、右孩子和学生数据：

#ifndef \_BINTREENODE\_H

#define \_BINTREENODE\_H

#include"student.h"

struct BinTreeNode{

BinTreeNode \*leftChild , \*rightChild;

Student data;

BinTreeNode(){

leftChild = rightChild = NULL;

}

BinTreeNode(Student &s , BinTreeNode \*temp = NULL , BinTreeNode \*temp2 = NULL){

data = s;

leftChild = temp;

rightChild = temp2;

}

};

#endif

二叉树类，只保留了部分代码，其他代码在附录。

#ifndef \_BINTREE\_H

#define \_BINTREE\_H

#include<fstream>

#include<stack>

#include"BinTreeNode.h"

class BinTree{

public:

//使用前序遍历建立二叉树 ,sig是代表空的标志

void createByFront(BinTreeNode \*&root , Student \*s , char \*sig , int &i);

void createByFront(Student \*s ,int n = 0){

char \*sig = "00000000000";

createByFront(root , s , sig , n);

}

private:

BinTreeNode \*root;

};

#endif

还有一个main函数，main函数是此程序的主要界面，是用户的主要交互界面，用户在此界面可以根据提示进行选择，选择不同的功能，再根据不同功能的不同操作来完成不同的任务，以此达到使用此系统的目的，具体代码在附录main函数部分。

实验设计

**采用的关键技术：**

程序采用的关键技术如下：

1.学生类（多种数据、对学生对象的基本操作）；

2.二叉树（建立、遍历、写入文件、将文件中的内容还原成二叉树等）；

3.类成员函数；

4.文件操作；

**主要设计思想与设计流程：**

设计思想：

程序是围绕学生健康管理系统来设计的，主要的设计思想就是建立基本的学生信息类，学生信息类中有学生的基本信息，也有对学生对象的基本操作。程序还利用了二叉树这种数据结构，二叉树是一种非线性表，操作上要比链表更加困难，但是二叉树所拥有的多种遍历方式、查找方式等又是链表所不能完成的，使用二叉树也因此有了多种优点。

设计流程：

学生类

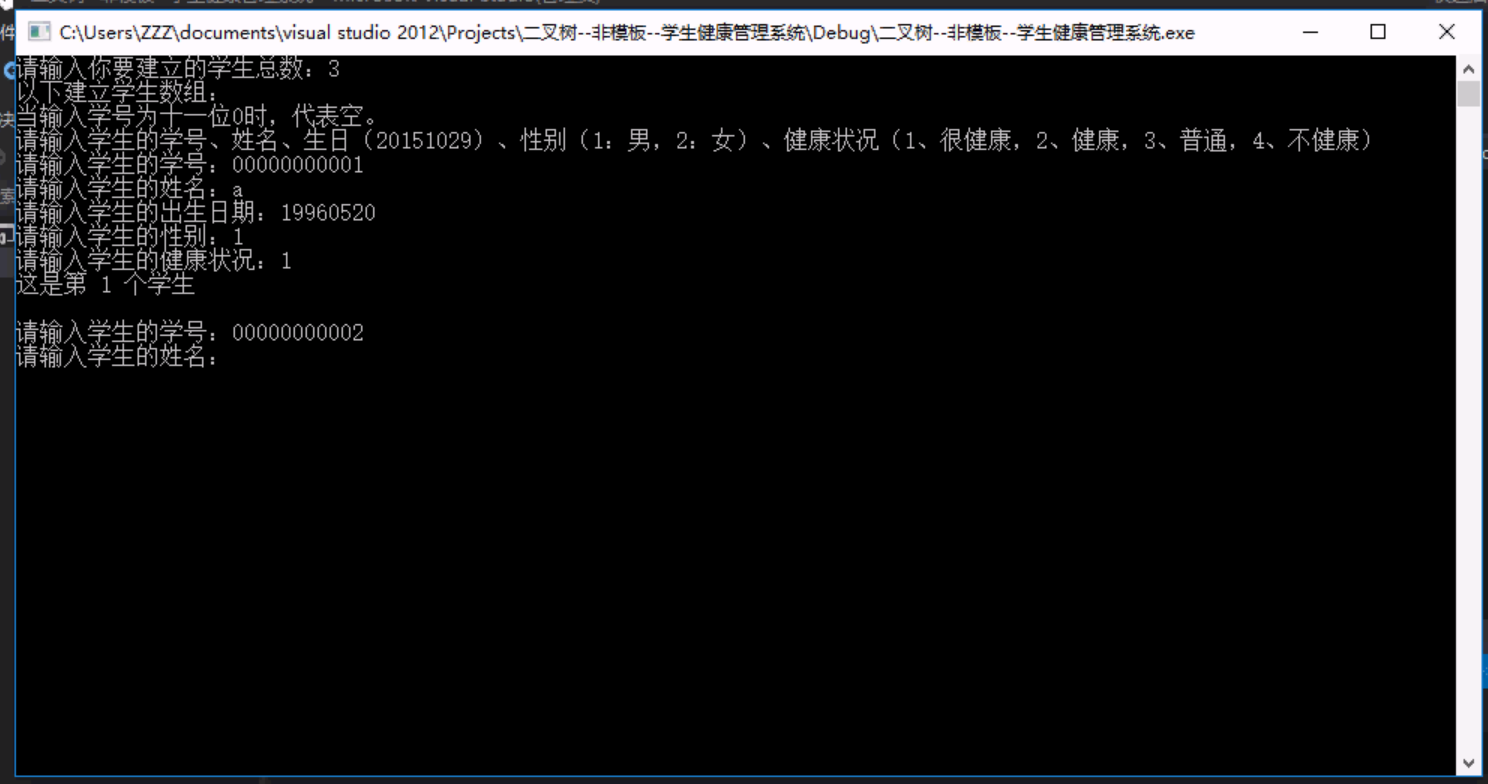
建立二叉树结点类

建立二叉树类

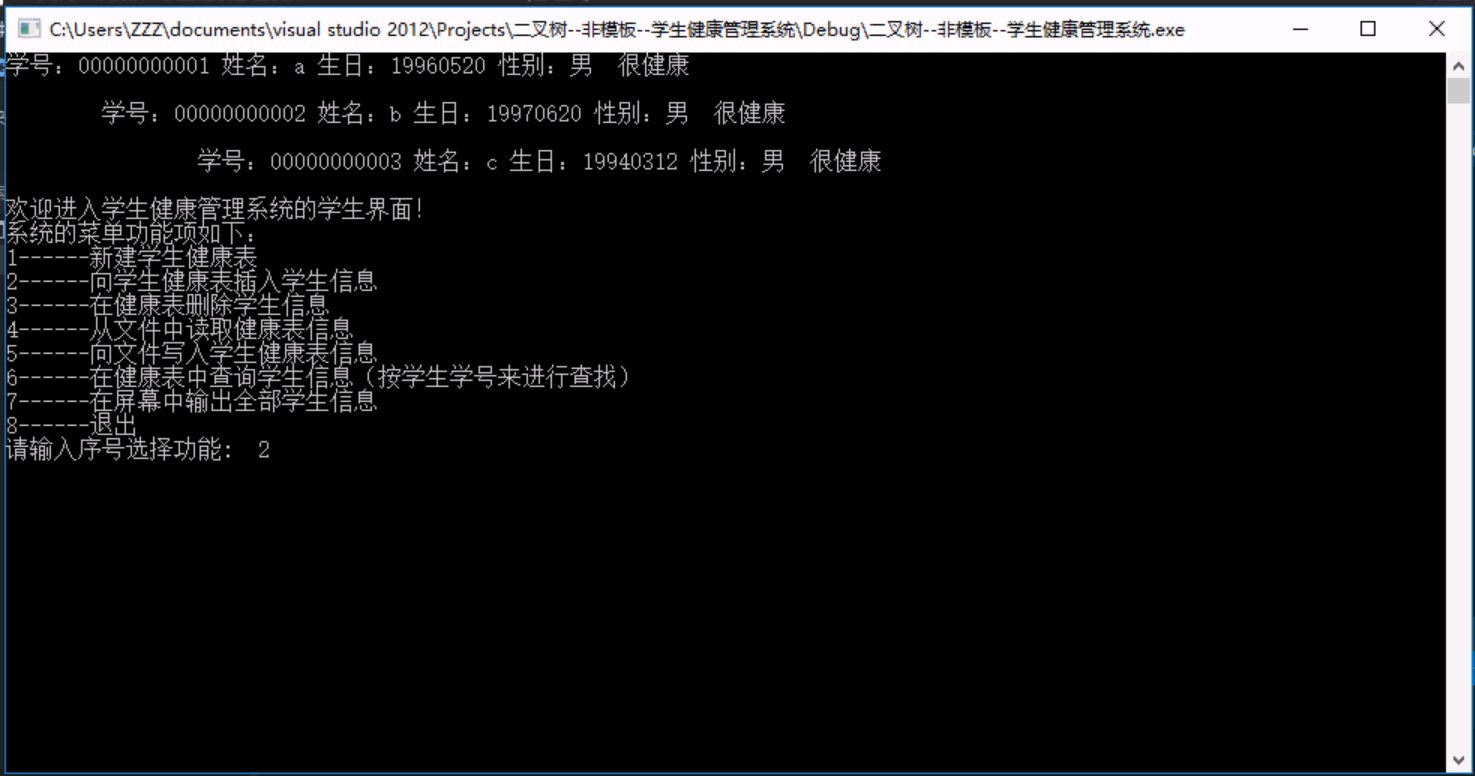
操作使用

实验实现

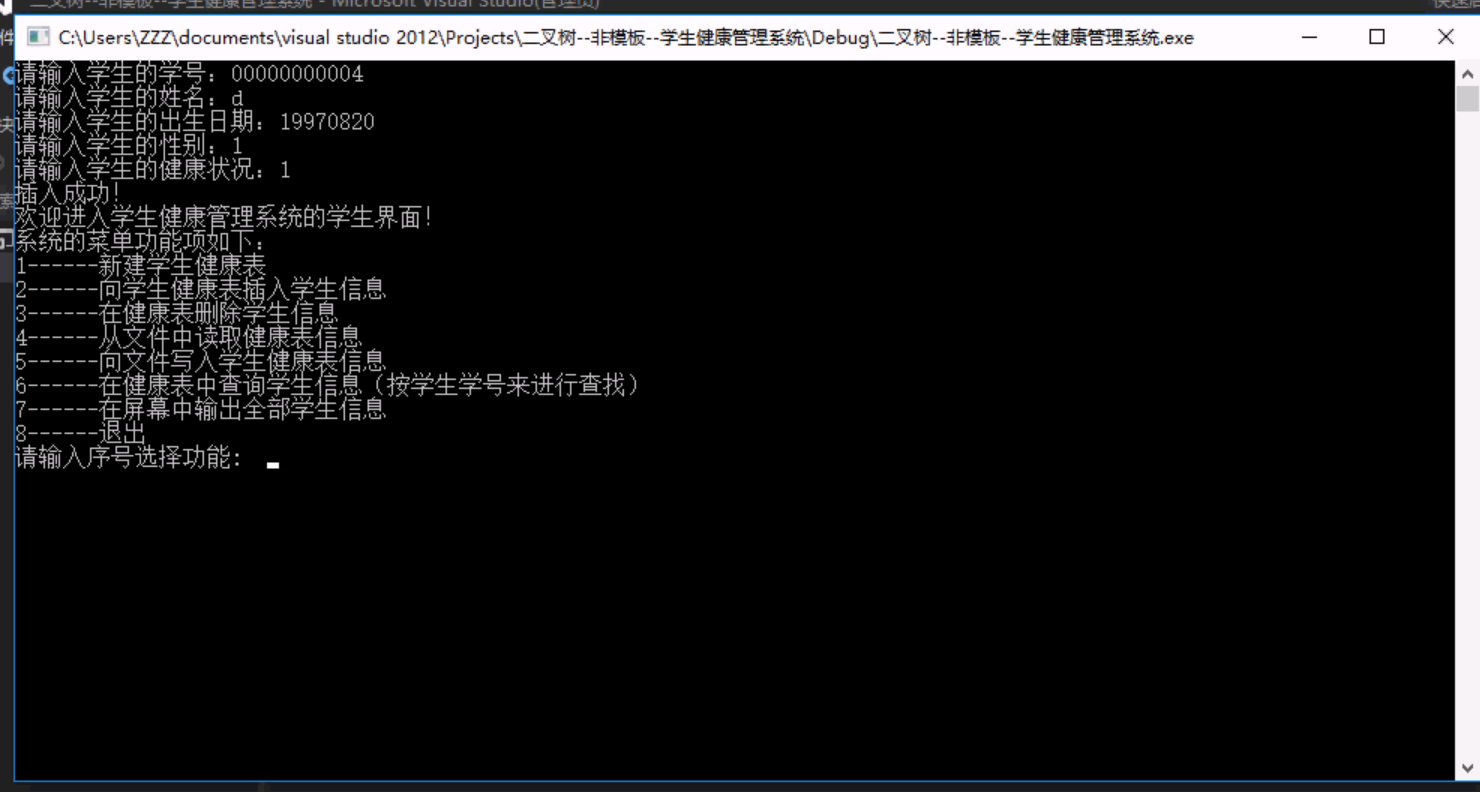
**主要功能模块分析:**

****

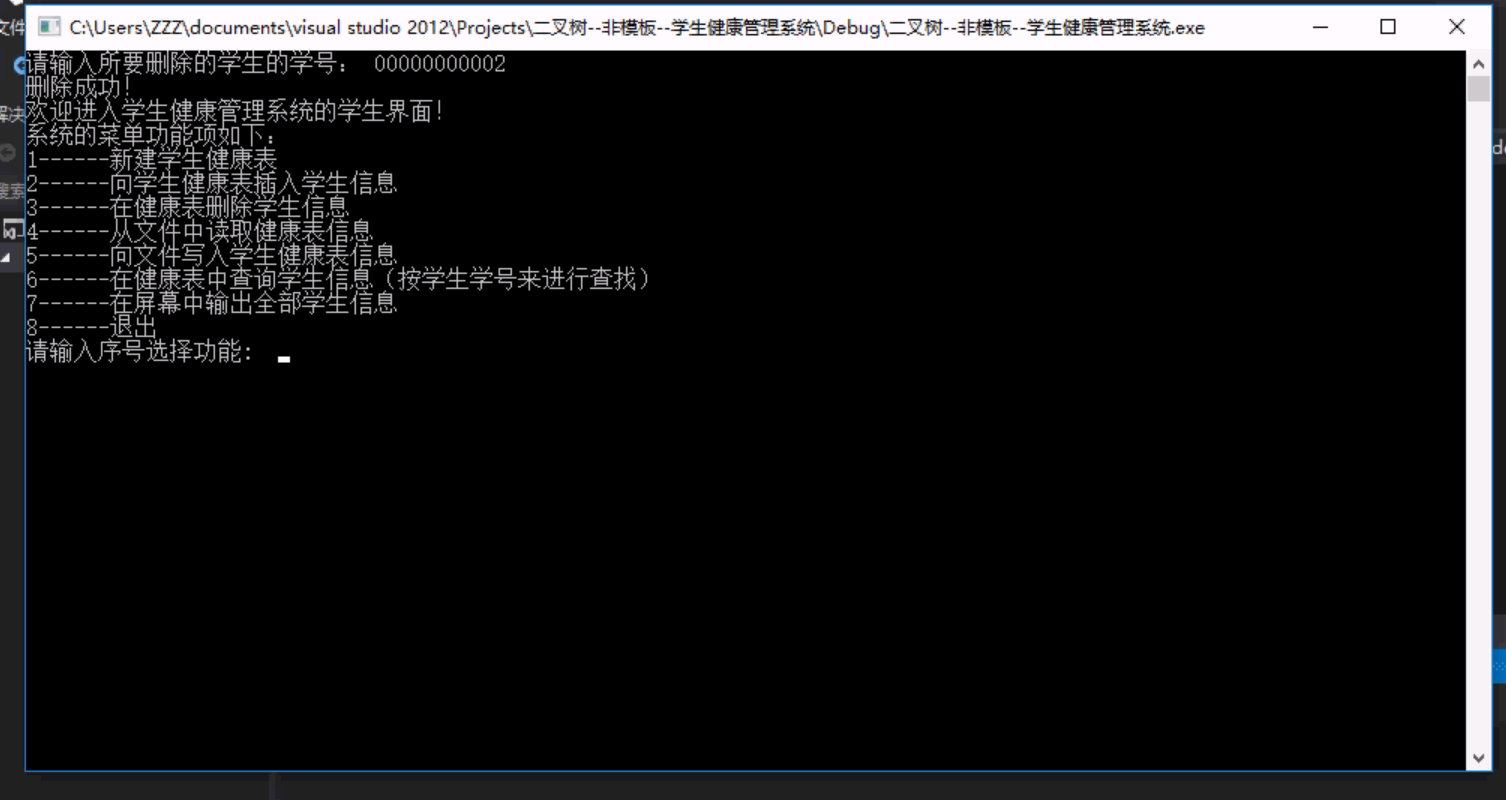
新建功能：新建功能会创建学生数组，创建的学生数组将用来创建二叉树，而且这是采用带空节点的前序遍历方式建立二叉树，所以需要输入学号为全0的代表是空二叉树。

****

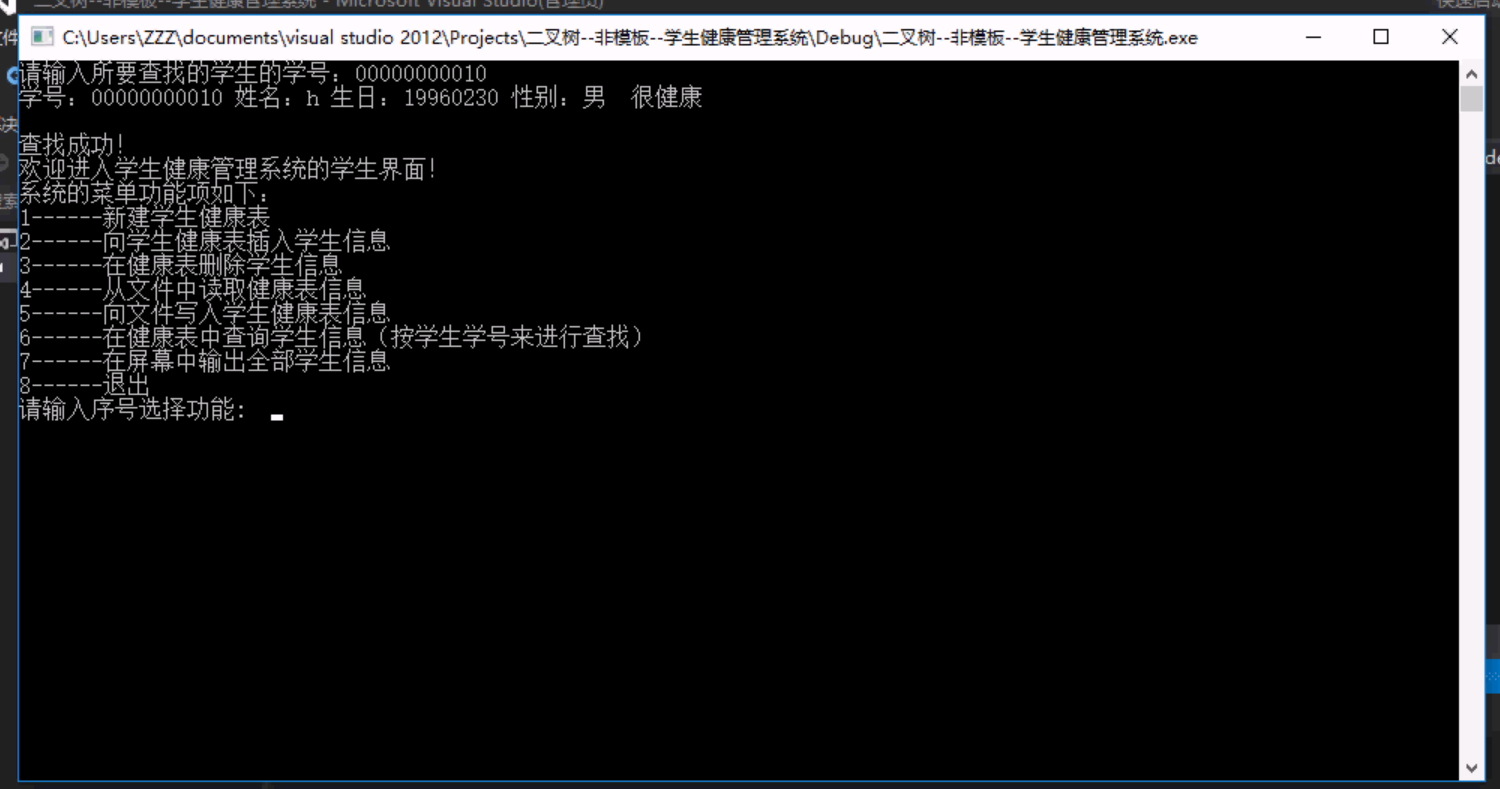
输出功能：输出功能按照层次输出，在同一列的代表是同一层，所以如果在第一列的有且只有一个数据。使用这种方式输出有一定的优势，例如输出的结果清晰明了的可以看出他们是在什么层次。

****

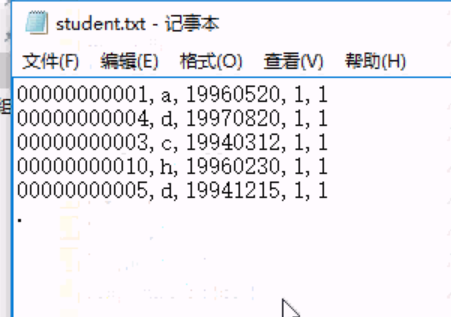
插入功能：插入学生数据

****

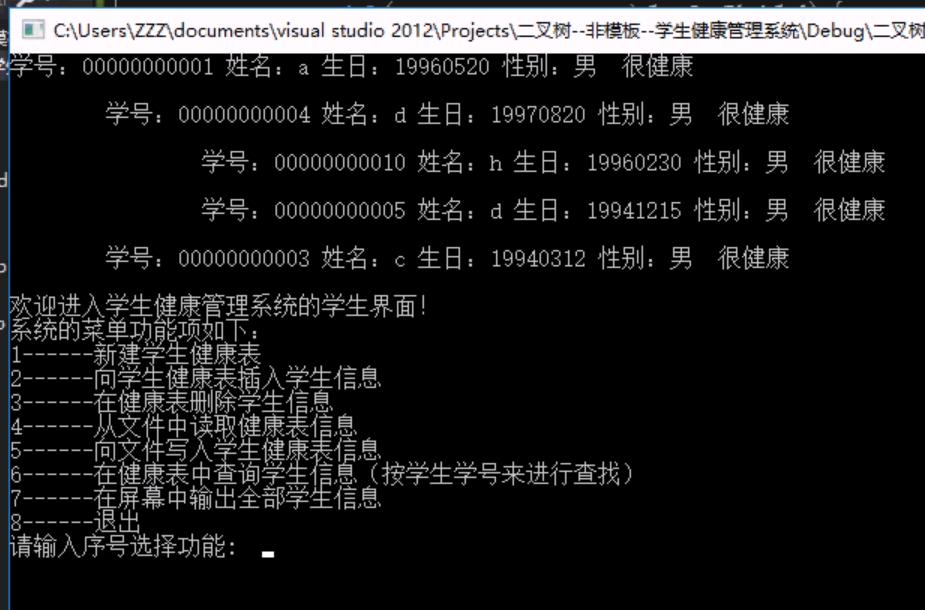
删除功能：使用学号查找删除的数据并删除。

****

查找功能：使用学号查找相应的学生信息并输出显示。

****

文件写入功能：把二叉树的内容写入到文件中，按着如图所示的格式写入，文件写入后可以在本地查看，在本地查看尽量不要修改文件的内容，否则是用二叉树进行读取时会出现问题。

****

文件读取功能：把之前写入到本地的文件进行读取出来，读取以后可以使用输出功能将学生信息输出查看。

**采用的软硬件平台介绍:**

软件平台：

Windows10：

美国[微软](http://baike.baidu.com/view/2353.htm" \t "http://baike.baidu.com/_blank)公司所研发的新一代跨平台及设备应用的[操作系统](http://baike.baidu.com/subview/880/4940471.htm" \t "http://baike.baidu.com/_blank)。

Visual studio 2012：

[Visual Studio](http://baike.baidu.com/view/28727.htm" \t "http://baike.baidu.com/item/_blank)是微软公司推出的开发环境。是最流行的[Windows](http://baike.baidu.com/view/4821.htm" \t "http://baike.baidu.com/item/_blank)平台应用程序开发环境。

硬件平台：

处理器：Intel Core i7-4710MQ

内存：8GB

显卡：Intel HD Graphics 4600

结果分析和讨论

在这个程序的开发中，完成了一个学生健康管理系统的开发，这个系统能够完成多种多样的功能，其中的学生类包含多种学生数据类型，包括学生学号、姓名、生日、性别及健康状况。使用该学生类可以保存多样数据，使用上方便。在程序的开发中还定义了二叉树的节点类，这个节点的操作很简单，只是作为整个二叉树系统的基础类，二叉树可以使用这个节点进行扩展。

程序还开发了二叉树类，二叉树类中完成了多种操作，包括遍历二叉树、求树高、求叶子节点数、输出二叉树等。同时，还针对学生健康管理系统进行了很多扩展，比如插入学生对象数据、删除学生对象（通过传入学生的学号进行查找删除）、文件读取写入和查询学生信息。二叉树类的开发中为了能够方便扩展，即使这个学生健康管理系统使用不到，但是还是对一个功能开发了多种应用方式，比如遍历就有很多种开发方式：前序，中序，后序递归遍历、前序，中序，后序非递归遍历、层次遍历等。

程序所完成的学生健康管理系统可以简单进行扩展，可以方便的扩展成其他的不同的系统，只需进行简单的修改和适配即可。综上所述，在此开发的学生健康管理系统通用性较强，功能较为强大，是很不错的系统。

参考文献

《C++程序设计（第2版）》谭浩强

《数据结构（用面向对象方法和C++语言描述）》

《算法导论》

《数据结构（C语言版）》

附录

学生：

头文件：

#ifndef \_STUDENT\_H

#define \_STUDENT\_H

#include<iostream>

#include<fstream>

using namespace std;

class Student{

private:

char Id[13];

char name[16];

char birth[10];

int sex;

int health;

public:

//构造函数

Student(char \*Id = NULL , char \*name = NULL , char \*birth = NULL , int sex = 0 , int health = 0);

Student(Student &t);

~Student();

//输出student对象

void display();

//重载输入、输出函数

friend istream& operator >> (istream &is , Student &t);

friend ostream& operator << (ostream &os , Student &t);

//写入文件，和从文件读取，希望能够使用传入的地址来进行读取写入

bool writeToFile(char \*address = "C:\\student.txt");

bool readFromFile(char \*address = "C:\\student.txt");

//重载 文件读取写入

friend ifstream& operator >> (ifstream &ifs , Student &t);

friend ofstream& operator << (ofstream &ofs , Student &t);

//重载 ==

friend bool operator == (Student &t1 , Student &t2);

//这个类的set函数和get函数

bool setId(char \*id);

bool setName(char \*name);

bool setBirth(char \*birth);

bool setSex(int sex);

bool setHealth(int health);

char\* getId();

char\* getName();

char\* getBirth();

int getSex();

int getHealth();

friend Student\* newStudent();

};

#endif

cpp实现文件：

#include<fstream>

#include<iostream>

#include"Student.h"

using namespace std;

Student::Student(char \*Id , char \*name , char \*birth , int sex , int health){

if(Id != NULL)

for(int i = 0 ; i < 12 ; i++ ){

this->Id[i] = Id[i];

}

// this->Id[11] = '\0';

if(name != NULL)

for(int i = 0 ; i < 16 ; i++ ){

this->name[i] = name[i];

}

if(birth != NULL)

for(int i = 0 ; i < 9 ; i++ ){

this->birth[i] = birth[i];

}

// this->birth[8] = '\0';

this->sex = sex;

this->health = health;

}

Student::Student(Student &t){

char\* id = t.getId();

char\* name = t.getName();

char\* birth = t.getBirth();

int sex = t.getSex();

int health = t.getHealth();

try{

for(int i = 0 ; i < 12 ; i++)

this->Id[i] = id[i];

for(int i = 0 ; i < 16 ; i++)

this->name[i] = name[i];

for(int i = 0 ; i < 9 ; i++)

this->birth[i] = birth[i];

this->sex = sex;

this->health = health;

}catch (char\*){

cerr<<"NULL!"<<endl;

}

}

Student::~Student(){/\*

delete Id;

delete name;

delete birth;\*/

}

void Student::display(){

cout<<"学号："<<Id<<" 姓名："<<name<<" 生日："<<birth<<" 性别：";

switch(sex){

case 1:

cout<<"男";

break;

case 2:

cout<<"女";

break;

default :

cout<<"无效值";

break;

}

switch(health){

case 1:

cout<<" 很健康"<<endl;

break;

case 2:

cout<<" 健康"<<endl;

break;

case 3:

cout<<" 普通"<<endl;

break;

case 4:

cout<<" 不健康"<<endl;

break;

default:

cout<<" 正常"<<endl;

break;

}

}

//这种方式太麻烦

istream & operator >> (istream &is , Student &t){

char ch;

int i;

cin.sync();

for(i = 0 ; i < 11 ; i++){

is.get(ch);

t.Id[i] = ch;

}

t.Id[i] = '\0';

is.get(ch);

for(;ch == ' ';)is.get(ch);

for(i = 0 ; i < 15 ; i++){

t.name[i] = ch;

is.get(ch);

if(ch >= '0' && ch <= '9' || ch == ' ')break;

}

t.name[i] = '\0';

for(;ch == ' ';)is.get(ch);

for(i = 0 ; i < 8 ; i++){

t.birth[i] = ch;

is.get(ch);

}

t.birth[i] = '\0';

for(;ch == ' ';)is.get(ch);

t.sex = ch - 48;

for(;ch == ' ';)is.get(ch);

t.health = ch - 48;

return is;

}

ostream & operator << (ostream &os , Student &t){

os<<"学号："<<t.getId()<<" 姓名："<<t.getName()<<" 生日："<<t.getBirth()<<" 性别：";

switch(t.getSex()){

case 1:

os<<"男";

break;

case 2:

os<<"女";

break;

default :

os<<"无效值";

break;

}

switch(t.getHealth()){

case 1:

os<<" 很健康"<<endl;

break;

case 2:

os<<" 健康"<<endl;

break;

case 3:

os<<" 普通"<<endl;

break;

case 4:

os<<" 不健康"<<endl;

break;

default:

os<<" 正常"<<endl;

break;

}

return os;

}

ifstream& operator >> (ifstream &ifs , Student &t){

if(!ifs)return ifs;

char ch;

char cch[16];

ifs.get(ch);

while(ch < 48 || ch > 57)ifs.get(ch);

for(int i = 0 ; ch != ',' && i < 12; i++){

cch[i] = ch;

ifs.get(ch);

}

t.setId(cch);

ifs.get(ch);

for(int i = 0 ; ch != ',' && i < 16 ; i++){

cch[i] = ch;

ifs.get(ch);

}

t.setName(cch);

ifs.get(ch);

for(int i = 0 ; ch != ',' && i < 9 ; i++){

cch[i] = ch;

ifs.get(ch);

}

t.setBirth(cch);

ifs.get(ch);

t.setSex(ch);

ifs.get(ch);ifs.get(ch);

t.setHealth(ch);

return ifs;

}

ofstream& operator << (ofstream &ofs , Student &t){

if(!ofs)return ofs;

ofs<<t.getId()<<","<<t.getName()<<","<<t.getBirth()<<","

<<t.getSex()<<","<<t.getHealth()<<"."<<endl;

return ofs;

}

bool Student::writeToFile(char \*address){

ofstream out(address);

if(!out)return false;

out<<this->getId()<<","<<this->getName()<<","<<this->getBirth()

<<","<<this->getSex()<<","<<this->getHealth()<<".";//类内部函数不用使用get函数

return true;

}

bool Student::readFromFile(char \*address){

ifstream in(address);

if(!in)return false;

char ch;

in.get(ch);

for(int i = 0 ; ch != ',' && i < 12; i++){

Id[i] = ch;

in.get(ch);

}

in.get(ch);

for(int i = 0 ; ch != ',' && i < 16 ; i++){

name[i] = ch;

in.get(ch);

}

in.get(ch);

for(int i = 0 ; ch != ',' && i < 9 ; i++){

birth[i] = ch;

in.get(ch);

}

in.get(ch);

sex = ch;

in.get(ch);in.get(ch);

health = ch;

return true;

}

bool operator == (Student &t1 , Student &t2){

char \*id1 = t1.getId();

char \*id2 = t2.getId();

int i;

for(i = 0 ; i < 12 ; i++){

if(id1[i] != id2[i])break;

}

if(i == 11)return true;

else return false;

}

//实现set方法

bool Student::setId(char \*id){

if(id == NULL)return false;

if(strlen(id) != 11)return false;

for(int i = 0 ; i < 12 ; i++ ){

this->Id[i] = id[i];

}

return true;

}

bool Student::setName(char \*name){

if(name == NULL)return false;

for(int i = 0 ; i < 16 ; i++){

this->name[i] = name[i];

}

return true;

}

bool Student::setBirth(char \*birth){

if(birth == NULL)return false;

if(strlen(birth) != 8)return false;

return true;

}

bool Student::setSex(int sex){

if(sex != 1 || sex != 2 )return false;

this->sex = sex;

return true;

}

bool Student::setHealth(int health){

if(health < 0 || health > 5)return false;

return true;

}

//实现get方法

char\* Student::getId(){

return Id;

}

char\* Student::getName(){

return name;

}

char\* Student::getBirth(){

return birth;

}

int Student::getSex(){

return sex;

}

int Student::getHealth(){

return health;

}

Student\* newStudent(){

char id[13];

char name[16];

char birth[10];

int sex ;

int health;

cin.sync();

cout<<"请输入学生的学号：";

cin>>id;

while(strlen(id) != 11){

cout<<"重新输入学号：";cin>>id;

}

if(strcmp(id , "00000000000") == 0){

Student \*student = new Student(id , "xx" , "12345678" , 1 , 1);

return student;

}

cout<<"请输入学生的姓名：";

cin.sync();

cin>>name;

cout<<"请输入学生的出生日期：";

cin>>birth;

while(strlen(birth) != 8){

cout<<"重新输入出生日期：";cin>>birth;

}

cout<<"请输入学生的性别：";

cin.sync();

cin>>sex;

while(sex != 1 && sex != 2){

cout<<"请重新输入学生的性别：";cin>>sex;

}

cout<<"请输入学生的健康状况：";

cin.sync();

cin>>health;

Student \*student = new Student(id , name , birth , sex , health);

return student;

}

二叉树节点：

头文件：

#ifndef \_BINTREENODE\_H

#define \_BINTREENODE\_H

#include"student.h"

struct BinTreeNode{

BinTreeNode \*leftChild , \*rightChild;

Student data;

BinTreeNode(){

leftChild = rightChild = NULL;

}

BinTreeNode(Student &s , BinTreeNode \*temp = NULL , BinTreeNode \*temp2 = NULL){

data = s;

leftChild = temp;

rightChild = temp2;

}

};

#endif

二叉树：

头文件：

#ifndef \_BINTREE\_H

#define \_BINTREE\_H

#include<fstream>

#include<stack>

#include"BinTreeNode.h"

class BinTree{

public:

BinTree(BinTreeNode \*temp = NULL){

root = temp;

}

~BinTree(){}

//使用前序遍历建立二叉树 ,sig是代表空的标志

void createByFront(BinTreeNode \*&root , Student \*s , char \*sig , int &i);

void createByFront(Student \*s ,int n = 0){

char \*sig = "00000000000";

createByFront(root , s , sig , n);

}

//递归 前序、中序、后序遍历

void visitByFrontCycle(BinTreeNode \*temp);

void visitByFrontCycle(){

visitByFrontCycle(root);

}

void visitByMiddleCycle(BinTreeNode \*temp);

void visitByMiddleCycle(){

visitByMiddleCycle(root);

}

void visitByLastCycle(BinTreeNode \*temp);

void visitByLastCycle(){

visitByLastCycle(root);

}

//非递归 前序、中序、后序遍历

void visitByFront();

void visitByMiddle();

void visitByLast();

//求树高

int getHeight(){

return getHeight(root);

}

int getHeight(BinTreeNode \*root);

//求叶子节点数

int countLeaf(){

int n = 0;

countLeaf(root , n);

return n;

}

void countLeaf(BinTreeNode \*root , int &n);

int count(){

int n = 0;

count(root , n);

return n;

}

void count(BinTreeNode \*root , int &n);

//输出二叉树

void display(){

int n = 0;

display(root , n);

}

void display(BinTreeNode \*root , int n);

//学生 相关操作

//插入

bool insert(BinTreeNode \*root , Student \*s);

bool insert(Student \*s){

return insert(root , s);

}

//删除

bool del(BinTreeNode \*&tree , char \*s , BinTreeNode \*temp);

bool del(char \*s){

BinTreeNode \*temp = NULL;

return del(root , s , temp);

}

//文件读取

bool readFromFile(BinTreeNode \*&root , char \*ad);

bool readFromFile(){

char \*a = "C:\\student.txt";

return readFromFile(root , a);

}

//文件写入

bool writeToFile(BinTreeNode \*root , char \*ad);

bool writeToFile(){

char \*a = "C:\\student.txt";

return writeToFile(root , a);

}

//查询

bool search(BinTreeNode \*root , char \*id);

bool search(char \*id){

return search(root , id);

}

private:

BinTreeNode \*root;

};

#endif

cpp实现文件：

#include"BinTree.h"

//使用前序遍历建立二叉树 ,sig是代表空的标志

void BinTree::createByFront(BinTreeNode \*&root , Student \*s , char \*sig , int &i){

if( !strcmp(s[i].getId() , "99999999999" )|| !strcmp(s[i-1].getId() , "99999999999"))return ;

if(strcmp(s[i].getId() , sig)){

root = new BinTreeNode(s[i]);

i++;

createByFront(root->leftChild , s , sig , i);

if( !strcmp(s[i].getId() , "99999999999" )|| !strcmp(s[i-1].getId() , "99999999999"))return ;

i++;

createByFront(root->rightChild , s , sig , i);

}else {

root = NULL;

}

}

//递归 前序、中序、后序遍历

void BinTree::visitByFrontCycle(BinTreeNode \*temp){

if(temp != NULL){

cout<<temp->data<<endl;

visitByFrontCycle(temp->leftChild);

visitByFrontCycle(temp->rightChild);

}

}

void BinTree::visitByMiddleCycle(BinTreeNode \*temp){

if(temp != NULL){

visitByFrontCycle(temp->leftChild);

cout<<temp->data<<endl;

visitByFrontCycle(temp->rightChild);

}

}

void BinTree::visitByLastCycle(BinTreeNode \*temp){

if(temp != NULL){

visitByFrontCycle(temp->leftChild);

visitByFrontCycle(temp->rightChild);

cout<<temp->data<<endl;

}

}

//非递归 前序、中序、后序遍历

void BinTree::visitByFront(){

stack<BinTreeNode \*> stu;

BinTreeNode \*temp = root;

stu.push(temp);

for( ; !stu.empty() ; ){

temp = stu.top();

stu.pop();

if(temp->rightChild != NULL)stu.push(temp->rightChild);

if(temp->leftChild != NULL)stu.push(temp->leftChild);

cout<<temp->data<<endl;

}

}

void BinTree::visitByMiddle(){

stack<BinTreeNode \*> stu;

BinTreeNode \*temp = root;

for(;;){

for(;temp != NULL;){

stu.push(temp = temp->leftChild);

}

do{

temp = stu.top();

stu.pop();

cout<<temp->data<<endl;

if(stu.empty())return;

}while(temp->rightChild == NULL || temp->rightChild == stu.top());

stu.push(temp->rightChild);

}

}

void BinTree::visitByLast(){

stack<BinTreeNode \*> stu;

BinTreeNode \*temp = root;

for(;;){

for(;temp != NULL;){

stu.push(temp = temp->leftChild);

}

do{

temp = stu.top();

stu.pop();

cout<<temp->data<<endl;

if(stu.empty())return;

}while(temp == stu.top()->rightChild || stu.top()->rightChild == NULL);

stu.push(temp->rightChild);

}

}

//求树高

int BinTree::getHeight(BinTreeNode \*root){

if(root != NULL){

int m = getHeight(root->leftChild);

int n = getHeight(root->rightChild);

return (m>=n)?m:n;

}

return 0;

}

//求叶子节点数

void BinTree::countLeaf(BinTreeNode \*root , int &n){

if(root != NULL){

if(root->leftChild == NULL && root->rightChild == NULL){

n++;

}else{

countLeaf(root->leftChild , n);

countLeaf(root->rightChild , n);

}

}

}

void BinTree::count(BinTreeNode \*root , int &n){

if(root != 0){

n++;

count(root->leftChild , n);

count(root->rightChild , n);

}

}

//输出二叉树

void BinTree::display(BinTreeNode \*root , int n){

if(root != NULL){

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

cout<<" ";

cout<<root->data;

cout<<endl;

n++;

display(root->leftChild , n);

display(root->rightChild , n);

n--;

}

}

bool BinTree::insert(BinTreeNode \*root , Student \*s){

if(root == NULL){

root = new BinTreeNode(\*s);

return true;

}

if(!strcmp(root->data.getId() , "00000000000")){

root = new BinTreeNode(\*s);

return true;

}

if(root->leftChild == NULL){

root->leftChild = new BinTreeNode(\*s);

return true;

}

if(root->rightChild == NULL){

root->rightChild = new BinTreeNode(\*s);

return true;

}

bool flag = insert(root->leftChild , s);

if(flag)return flag;

bool flag2 = insert(root->rightChild , s);

if(flag2)return flag2;

return false;

}

bool BinTree::del(BinTreeNode \*&tree , char \*s , BinTreeNode \*temp){

if(tree == NULL)return false;

if(!strcmp(tree->data.getId() , s)){

if(temp == NULL){

BinTreeNode \*t = tree->leftChild;

temp = tree->rightChild;

tree = t;

for(; t->leftChild != NULL ; t = t->leftChild);

t->leftChild = temp;

return true;

}

if(tree->leftChild == NULL && tree->rightChild == NULL){

if(tree == temp->leftChild){

temp->leftChild = NULL;

return true;

}

if(tree == temp->rightChild){

temp->rightChild = NULL;

return true;

}

}

if(tree == temp->leftChild){

BinTreeNode \*t = tree->rightChild;

tree = tree->leftChild;

for(; temp->leftChild != NULL ; temp = temp->leftChild);

temp->leftChild = t;

return true;

}

if(tree == temp->rightChild){

BinTreeNode \*t = tree->leftChild;

tree = tree->rightChild;

for(; temp->rightChild != NULL ; temp = temp->rightChild);

temp->rightChild = t;

return true;

}

}else {

temp = root;

if(del(tree->leftChild , s , temp))return true;

if(del(tree->rightChild , s , temp))return true;

}

return false;

}

bool BinTree::readFromFile(BinTreeNode \*&root , char \*ad){

ifstream in(ad);

if(!in)

return false;

char Id[13];

char name[16];

char birth[10];

int sex;

int health;

char ch;

Student \*student;

BinTreeNode \*temp;

for(; !in.eof() ;){

in.get(ch);

for( ; ch == '\n' ;)in.get(ch);

if(ch == '.')break;

for(int i = 0 ; ch != ',' && i < 12; i++){

Id[i] = ch;

in.get(ch);

}

Id[11] = '\0';

in.get(ch);

for(int i = 0 ; ch != ',' && i < 16 ; i++){

name[i] = ch;

in.get(ch);

if(ch == ',')name[i+1] = '\0';

}

in.get(ch);

for(int i = 0 ; ch != ',' && i < 9 ; i++){

birth[i] = ch;

in.get(ch);

}

birth[8] = '\0';

in.get(ch);

sex = ch-48;

in.get(ch);in.get(ch);

health = ch-48;

student = new Student(Id , name , birth , sex , health);

if(root == NULL)root = temp = new BinTreeNode(\*student);

else if(temp->leftChild == NULL)temp->leftChild = new BinTreeNode(\*student);

else if(temp->rightChild == NULL)temp->rightChild = new BinTreeNode(\*student);

else

insert(student);

}

return true;

}

bool BinTree::writeToFile(BinTreeNode \*root , char \*ad){

ofstream out(ad);

if(!out)return false;

stack<BinTreeNode \*>s;

BinTreeNode \*temp =root;

s.push(temp);

for( ; !s.empty() ; ){

temp = s.top();

s.pop();

out<<temp->data.getId()<<","<<temp->data.getName()<<","<<temp->data.getBirth()

<<","<<temp->data.getSex()<<","<<temp->data.getHealth()<<endl;

if(temp->leftChild != NULL)s.push(temp->leftChild);

if(temp->rightChild != NULL)s.push(temp->rightChild);

}

out<<".";

return true;

}

bool BinTree::search(BinTreeNode \*root , char \*id){

if(root == NULL)

return false;

if(!strcmp(root->data.getId() , id)){

cout<<root->data<<endl;

return true;

}

bool flag = search(root->leftChild , id);

if(flag)return flag;

flag = search(root->rightChild , id);

return flag;

}

main函数：

#include<vector>

#include"BinTree.h"

int main(){

BinTree \*tree = new BinTree();

for(;;){

cout<<"欢迎进入学生健康管理系统的学生界面！"<<endl;

cout<<"系统的菜单功能项如下："<<endl;

cout<<"1------新建学生健康表 "<<endl;

cout<<"2------向学生健康表插入学生信息 "<<endl;

cout<<"3------在健康表删除学生信息 "<<endl;

cout<<"4------从文件中读取健康表信息 "<<endl;

cout<<"5------向文件写入学生健康表信息 "<<endl;

cout<<"6------在健康表中查询学生信息（按学生学号来进行查找） "<<endl;

cout<<"7------在屏幕中输出全部学生信息 "<<endl;

cout<<"8------退出 "<<endl;

cout<<"请输入序号选择功能: ";

int i;

cin>>i;

if(i)system("cls");

if(i == 1){

int n;

cout<<"请输入你要建立的学生总数：";

cin>>n;

if(n > 0){

Student \*stu = new Student[n+1];

cout<<"以下建立学生数组："<<endl;

cout<<"当输入学号为十一位0时，代表空。"<<endl;

cout<<"请输入学生的学号、姓名、生日（20151029）、"

"性别（1：男，2：女）、健康状况（1、很健康，2、健康，3、普通，4、不健康）"<<endl;

int k;

for(k = 0 ; k < n ; k++){

stu[k] = \*newStudent();

cout<<"这是第 "<<k+1<<" 个学生\n"<<endl;

}

if(stu[k].setId("99999999999"))

tree->createByFront(stu);

else {

cout<<"WRONG！"<<endl;

}

}

}

if(i == 2){//插入

Student \*stu;

stu = newStudent();

if(tree->insert(stu))

cout<<"插入成功！"<<endl;

else

cout<<"插入失败！"<<endl;

}

if(i == 3){//删除

char Id[12];

cout<<"请输入所要删除的学生的学号： ";

cin>>Id;

if(tree->del(Id)){

cout<<"删除成功！"<<endl;

}

else

cout<<"删除失败！"<<endl;

}

if(i == 4){//文件读取

cout<<"开始文件读取："<<endl;

if(tree->readFromFile())

cout<<"读取成功！"<<endl;

else

cout<<"读取失败！"<<endl;

}

if(i == 5){//文件写入

cout<<"开始文件写入："<<endl;

if(tree->writeToFile())

cout<<"写入成功！"<<endl;

else

cout<<"写入失败！"<<endl;

}

if(i == 6){//查询，学号

char Id[12];

cout<<"请输入所要查找的学生的学号：";

cin>>Id;

if(tree->search(Id))

cout<<"查找成功！"<<endl;

else

cout<<"查找失败！"<<endl;

}

if(i == 7){//输出

tree->display();

}

if(i == 8)return 0;

if(i >= 9)continue;

}

system("pause");

return 0;

}