华南师范大学本科生实验报告

姓名 邓智超　　学号 20192121026

院系 计算机学院 专业 计算机科学与技术（师范）

年级 19 级　　班级 1 班

实验时间 2020 年 11 月 27 日

实验名称 二叉树

课程名称 数据结构实验

华南师范大学教务处编印

华南师范大学本科生实验报告

实验课程：数据结构实验课

实验名称：二叉树

一、实验内容

实现学生通讯录管理的几个操作功能（新建、插入、删除、从文件读取、写入文件和查询、屏幕输出等功能）。通讯录中学生的信息有学号、姓名、出生日期、性别、电话和地址等。

- **必做内容**

◆利用二叉链式存储结构来实现；

◆系统的菜单功能项如下：

1------新建学生通讯录（以层次遍历的方式进行学生信息的输入）
2------向学生通讯录插入学生信息
3------在通讯录中查询学生信息（按学号查询）
4------在通讯录中查询学生信息（按姓名查询）
5------在通讯录中查询学生信息（按电话号码查询）
6------在通讯录删除学生信息（先按上述 3-5 定位被删除学生）
7------输出年龄最小的学生信息
8------在屏幕中输出全部学生信息（可以选择遍历的方式）
9------从文件中读取通讯录信息
10------向文件写入学生通讯录信息
11-----退出

◆生成一个达百万个学生数据信息的通讯录文件，并让该系统管理这个学生通讯录。

◆如果如果学生人数超过千万的时候，会出现什么现象，你认为该如何解决？最后把上述问题的解决方案写入到实验文档中。

◆分析系统的整体性性能，并思考原因以及解决方案。最后把相应的研究分析报告书写入到实验文档中。

选做内容

用 MFC 的单文档窗口和菜单设计界面。

- **已完成内容**

必做部分：

✓ 在 Visual Studio 平台使用 c++语言完成系统的所有功能，并且对用户交互界面做了一定的优化，使得控制台界面看起来更加简洁舒适。所有功能可正常使用且具有一定的健壮性。

- ✓ 按照要求生成了一个达到百万学生数据的通讯录文件（存放在 randomData.dat 中），可以在系统中读入并管理这个通讯录文件。
- ✓ 提出了解决学生人数超过千万时系统速度变慢的问题，并完成了分析系统性能的研究分析报告书（详见实验文档）

  具体实现过程请查看实验文档部分。

选做部分：

- ✓ 已通过 QT 平台完成了窗口和菜单设计界面，实现了图形化的系统。按照需求，该系统可以实现创建通讯录，添加，删除，查找，输出，读取，写入，读入百万数据的功能。

二、实验目的

- 加深对二叉树的遍历操作及实现算法的理解，以便在解决实际问题中灵活运用它们。
- 熟悉利用递归的方法编写对二叉树这种递归数据结构进行处理的算法。
- 熟悉利用非递归方法编写对二叉树进行遍历应用的算法。

三、实验文档：

必做部分：

实现思路

学生通讯录管理系统共包括 student.h 、 studentDataBase.h studentDataBase.cpp、randomData.cpp、main.cpp 五个文件。

student 结构用于存放单个学生对象的信息，以及对运算符的重载（输入，输出，赋值运算符）。

定义了三个辅助结构 nameTreeNode，numberTreeNode，phoneNumTreeNode，分别用于构成以姓名、学号、电话为关键字的二叉搜索树作为辅助树，进行搜索时，先通过搜索树找到目标结点的地址，通过访问地址去访问目标结点。以 nameTreeNode 为例，它包括一个字符数组和一个指向 student 的指针，以及 leftChild 和 rightChild 指针。它将以学生的姓名为关键字构建出一个二叉搜索树作为辅助结构，进行查找时，先通过二叉搜索树，取出目标关键字所在结点，通过访问该结点的 student 指针就可以获得目标结点的地址，之后访问目标节点地址即可。

studentDataBase 类用于构建二叉树以及各种对链表的操作实现系统的功能（添加，删除等等），建立二叉树时，以层次序遍历的方式建立。

main 函数中使用了函数指针数组来代替 switch，提升了效率。用于显示菜单、接受用户指令并调用对应的函数。

randomData.cpp 文件用于随机产生百万数据并存储在 randomData.dat 文件中。

全局变量声明

```
//声明全局类对象
studentDataBase studata;
nameTreeNode *nameTree=0;//以姓名为关键字的搜索树
numberTreeNode* numberTree = 0;//以学号为关键字的搜索树
phoneNumTreeNode* phoneNumTree = 0;//以电话为关键字的搜索树
student* MIN = 0;//保存年龄最小的学生地址
```

Student 结构体声明

数据成员包括 name(姓名)，number(学号)，gender(性别)，bitrthday(出生日期)、phoneNumber(电话号码)、address（地址），这六个属性都声明为字符数组。还有一个用于标记是否被删除的标记 isDeleted，若值为 0，则没有被删除，若值为 1，表示被删除。

方法包括重载<<运算符，重载>>运算符，重载=运算符。实现方法较为简单，此处不做详细叙述。

重载<<运算符

函数原型：

```
friend ostream& operator<<(ostream& out, student stu)//重载输出
```

实现方式：

此函数的功能是对 studen 结构体重载<<运算符。按照一定的字长，依次输出 student 类对象的 number，name，birthday，gender，phoneNumber，address 属性。

代码：

```
//返回值：out
//函数功能：重载输出函数，方便输出学生信息
friend ostream& operator<<(ostream& out, student stu)//重载输出
{
    out << setw(15) << stu.number;
    out << setw(10) << stu.name;
    out << setw(12) << stu.birthday;
    out << setw(8) << stu.gender;
    out << setw(15) << stu.phoneNumber;
    out << setw(20) << stu.address;
    return out;
}
```

重载>>运算符

函数原型：

```
friend istream& operator>>(istream& in, student &stu)//重载输入
```

实现方式：

此函数的功能是对 studen 结构体重载>>运算符。按照顺序依次输入 student 类对象的 number，name，birthday，gender，phoneNumber，address 属性即可。若输入的 number 第一个字符是#，说明输入已经结束了，就不需要输入其他五个属性了。

代码：

```
//返回值：in
//函数功能：重载输入函数，方便输入学生信息
friend istream& operator>>(istream& in, student& stu)//重载输入
{
    in >> stu.number;
    if (stu.number[0] == '#')//表示输入结束
        return in;
    else
        in >> stu.name >> stu.birthday >> stu.gender >> stu.phoneNumber >> stu.address;
    return in;
}
```

重载=运算符

函数原型：

```
void operator=(student* p2)//重载=
```

实现方式：

此函数的功能是对 studen 结构体重载>=运算符。按照顺序依次将目标 student 类对象的 number，name，birthday，gender，phoneNumber，address 属性赋值给待赋值的对象即可。

代码：

```
    //返回值：无
    //函数功能：重载=操作符，方便赋值
    void operator=(student* p2)//重载=
    {
        this->number = p2->number;
        this->name = p2->name;
        this->birthday = p2->birthday;
        this->gender = p2->gender;
        this->phoneNumber = p2->phoneNumber;
        this->address = p2->address;
    }
```

student 类声明代码和实现如下：

```
#pragma once
#include<iostream>
#include<string>
#include<iomanip>
using namespace std;
struct student
{
    //学号、姓名、出生日期、性别、电话和地址
    char number[15];//学号
    char name[10];//姓名
    char birthday[12];//出生日期
    char gender[8];//性别
    char phoneNumber[15];//电话
    char address[20];//地址
    student* leftChild = NULL;
    student* rightChild = NULL;
```

```
    bool isDeleted=0;//判断是否被删除 若已经删除，则 isDeleted 值为 1
    friend ostream& operator<<(ostream& out, student stu)//重载输出
    {
        out << setw(15) << stu.number;
        out << setw(10) << stu.name;
        out << setw(12) << stu.birthday;
        out << setw(8) << stu.gender;
        out << setw(15) << stu.phoneNumber;
        out << setw(20) << stu.address;
        return out;
    }
    friend istream& operator>>(istream& in, student& stu)//重载输入
    {
        in >> stu.number;
        if (stu.number[0] == '#')//表示输入结束
            return in;
        else
            in >> stu.name >> stu.birthday >> stu.gender >> stu.phoneNumber >> stu.address;
        return in;
    }
    void operator=(student* p2)//重载=
    {
        strcpy_s(this->number, p2->number);
        strcpy_s(this->name, p2->name);
        strcpy_s(this->birthday, p2->birthday);
        strcpy_s(this->gender, p2->gender);
        strcpy_s(this->phoneNumber, p2->phoneNumber);
        strcpy_s(this->address, p2->address);
    }
};
```

三个辅助结构体声明

nameTreeNode：

该结构用于构成以学生姓名为关键字的二叉搜索树作为辅助结构。其数据成员包括指向 student 的指针 pointer，用于存放姓名的字符数组，以及指向下一节点的指针。

代码：

```
//以姓名构建的搜索二叉树结点
//用于构建辅助结构
struct nameTreeNode
{
```

```
    char name[10];//姓名
    student* pointer = 0;//指向二叉树中的结点
    nameTreeNode* leftChild = 0;
    nameTreeNode* rightChild = 0;
};
```

numberTreeNode：

该结构用于构成以学生学号为关键字的二叉搜索树作为辅助结构。其数据成员包括指向 student 的指针 pointer，用于存放学号的字符数组，以及指向下一节点的指针。

代码：

```
//以学号构建的搜索二叉树结点
//用于构建辅助结构
struct numberTreeNode
{
    char number[15];//学号
    student* pointer = 0;//指向二叉树中的结点
    numberTreeNode* leftChild = 0;
    numberTreeNode* rightChild = 0;
};
```

phoneNumTreeNode:

该结构用于构成以学生电话为关键字的二叉搜索树作为辅助结构。其数据成员包括指向 student 的指针 pointer，用于存放电话的字符数组，以及指向下一节点的指针。

代码：

```
//以电话号码构建的搜索二叉树结点
//用于构建辅助结构
struct phoneNumTreeNode
{
    char phoneNumber[15];//电话号码
    student* pointer = 0;//指向二叉树中的结点
    phoneNumTreeNode* leftChild = 0;
    phoneNumTreeNode* rightChild = 0;
};
```

studentDatBase 类声明

studentDataBase 类用于将学生的通讯录信息构建成二叉树，以及进行各种对二叉树的操作。私有方法包括：前序、中序、后序遍历输出二叉树结点内容的函数

previsit()，midvisit()，behvisit()。公有成员包括保存根节点地址的指针 root，构造函数 studentDataBase()，建立通讯录函数 createTree()，按姓名查找函数 searchName()，按学号查找函数 searchNum()，按电话号码查找函数 searchPhoneNum()，按学号删除函数 delByNum()，按姓名删除函数 delByName()，按电话号码删除函数 delByPhoneNum()，输出年龄最小学生的信息函数 getYoungest()，输出所有信息函数 output()，写入文件函数 writeFile()，读取文件函数 readFile()函数，插入函数 insert()，以姓名为关键字插入搜索树函数 insertByName()，以学号为关键字插入搜索树函数 insertByNum()，以电话为关键字插入搜索树函数 insertByPhoneNum()，释放四棵树的空间函数 deleteTree()函数。在接下来的内容会逐个说明每个函数的实现方法。

studentData 类声明代码如下：

```
#pragma once
#ifndef STUDENTDATABASE_H_
#define STUDENTDATABASE_H_
#include<string>
#include<iostream>
#include"student.h"
using namespace std;
class studentDataBase
{
private:
    void previsit(student* root);//前序输出
    void midvisit(student* root);//中序输出
    void behvisit(student* root);//后序输出
public:
    student* root;//根结点
    studentDataBase();//构造函数
     void createTree();//新建
    student* searchNum(numberTreeNode*root,char num[]);//按学号查找
    student* searchName(nameTreeNode*root,char name[]);//按姓名查找
    student* searchPhoneNum(phoneNumTreeNode*root,char phoneNum[]);//按电话查询
    bool delByNum(char num[]);//按学号删除
    bool delByName(char name[]);//按姓名删除
    bool delByPhoneNum(char phoneNum[]);//按电话删除
    bool insertByNum(student* p, numberTreeNode*& root1);//在学号数插入
    bool insertByName(student* p, nameTreeNode*& root1);//在姓名树插入
    bool insertByPhoneNum(student* p, phoneNumTreeNode*& root1);//在电话树插入
```

```
    void insert(student* p, student*& root);//在二叉树插入
    void getYoungest(student*root);//输出年龄最小学生信息
    void output();//输出学生信息
    void readFile(string fileName);//读入
    void writeFile(string fileName);//写入
    void deleteTree(student* root);//释放四棵树的空间
};
#endif
```

studentDataBase 类实现

接下来将逐个介绍 studentDataBase(),createTree(),searchName(),searchNum(),searchPhoneNum(),delByNum(),delByName(),delByPhoneNum(),getYoungest(),output(),writeFile(),readFile(),deleteTree(),insert() ,insertByName(),insertByNum(),insertByPhoneNum(),的实现方法(根据需要会适度调整说明顺序，并不是严格按照以上顺序的)。

在 main.cpp 声明了几个全局变量在 studentDataBase 的实现中会用到，在这里先作说明：

```
extern void title();//输出标题函数
extern studentDataBase studata;//二叉树类 类对象
extern nameTreeNode* nameTree;//姓名树根结点
extern numberTreeNode* numberTree;//学号树根结点
extern phoneNumTreeNode* phoneNumTree;//电话树根结点
extern student* MIN;//存储年龄最小的学生结点地址
```

deleteNameTree()

函数原型：

```
    void deleteNameTree(nameTreeNode* root)
```

返回值：

无

函数功能：

释放辅助结构（以姓名为关键字的二叉搜索树）的空间

函数参数：

以姓名为关键字的二叉搜索树的根结点地址 nameTree

实现方式：

采用后续遍历的方式遍历二叉搜索树，对每一个结点，先释放其左子树结点的空间，再释放其右子树结点的空间，最后释放根结点地址的空间。

代码：

```
//返回值：无
//函数功能：释放姓名树的空间
//函数参数：姓名树的根地址
void deleteNameTree(nameTreeNode* root)
{
    if (root == 0)return;
    else
    {
        deleteNameTree(root->leftChild);
        deleteNameTree(root->rightChild);
        delete root;
    }
}
```

deleteNumberTree()

函数原型：

```
void deleteNumberTree(numberTreeNode* root)
```

返回值：

无

函数功能：

释放辅助结构（以学号为关键字的二叉搜索树）的空间

函数参数：

以学号为关键字的二叉搜索树的根结点地址 numberTree

实现方式：

采用后续遍历的方式遍历二叉搜索树，对每一个结点，先释放其左子树结点的

空间，再释放其右子树结点的空间，最后释放根结点地址的空间。

代码：

```
//返回值：无
//函数功能：释放学号树的空间
//函数参数：学号树的根地址
void deleteNumberTree(numberTreeNode* root)
{
    if (root == 0)return;
    else
    {
        deleteNumberTree(root->leftChild);
        deleteNumberTree(root->rightChild);
        delete root;
    }
}
```

deletePhoneNumTree()

函数原型：

```
void deletePhoneNumTree(phoneNumTreeNode* root)
```

返回值：

无

函数功能：

释放辅助结构（以电话为关键字的二叉搜索树）的空间

函数参数：

以电话为关键字的二叉搜索树的根结点地址 phoneNumTree

实现方式：

采用后续遍历的方式遍历二叉搜索树，对每一个结点，先释放其左子树结点的空间，再释放其右子树结点的空间，最后释放根结点地址的空间。

代码：

```
//返回值：无
//函数功能：释放电话树的空间
//函数参数：电话树的根地址
void deletePhoneNumTree(phoneNumTreeNode* root)
```

```
{
    if (root == 0)return;
    else
    {
        deletePhoneNumTree(root->leftChild);
        deletePhoneNumTree(root->rightChild);
        delete root;
    }
}
```

deleteTree()

函数原型：

```
void studentDataBase::deleteTree(student* root)
```

返回值：

无

函数功能：

释放主体结构（以层次序遍历建立的二叉树）的空间

函数参数：

以层次序遍历建立的二叉树的根结点地址 root

实现方式：

采用后续遍历的方式遍历二叉树，对每一个结点，先释放其左子树结点的空间，再释放其右子树结点的空间，最后释放根结点地址的空间。

代码：

```
//返回值：无
//函数功能：释放二叉树的空间
//函数参数：二叉树的根地址
void studentDataBase::deleteTree(student* root)
{
    if (root == 0)return;
    else
    {
        deleteTree(root->leftChild);
        deleteTree(root->rightChild);
        delete root;
```

```
    }
}
```

deleteTrees()

函数原型：

```
    void deleteTrees()
```

返回值：

无

函数功能：

释放主体结构（以层次序遍历建立的二叉树）和辅助结构的空间（三棵二叉搜索树）的空间

函数参数：

无

实现方式：

依次调用上述的四个函数即可。

代码：

```
//返回值：无
//函数功能：释放上述四棵树的空间
//          一棵主树和三棵辅助树
void deleteTrees()
{
    studata.deleteTree(studata.root);
    deleteNameTree(nameTree);
    deleteNumberTree(numberTree);
    deletePhoneNumTree(phoneNumTree);
}
```

studentDataBase()

函数原型：

studentDataBase::studentDataBase()

返回值：

无

函数功能：

构造函数，将 root 初始化为 0

实现方式：

将 root 赋值为 0 即可

代码：

```
studentDataBase::studentDataBase()//构造函数
{
    root = 0;
}
```

insert()

函数原型：

```
void studentDataBase::insert(student* p, student* &root)
```

返回值：

无

函数功能：

以层次序遍历的方式，将目标结点插入到二叉树当中。

函数参数：

前者为目标结点地址，后者为二叉树的根地址。

实现方式：

首先按照层次序遍历的方式找到插入点(就是第一个 leftChild 或 rightChild 为 0 的结点)，之后将新的结点作为该结点的孩子插入即可

代码：

```
//返回值：无
//函数功能：以层次序遍历的方式，将目标结点插入到二叉树当中
//函数参数：前者为目标结点地址，后者为二叉树的根地址
void studentDataBase::insert(student* p, student* &root)
{
```

```
    if (root == 0)
        root = p;
    else
    {
        queue<student*>stu;
        stu.push(root);
        student* temp;
        while (!stu.empty())
        {
            temp = stu.front();
            stu.pop();
            if (temp->leftChild != 0)
                stu.push(temp->leftChild);
            else//说明将在此结点的左子树插入
            {
                temp->leftChild = p;
                break;
            }
            if (temp->rightChild != 0)
                stu.push(temp->rightChild);
            else//说明将在此结点的右子树插入
            {
                temp->rightChild = p;
                break;
            }
        }
    }
}
```

insertByName()

函数原型：

```
bool studentDataBase::insertByName(student* p, nameTreeNode*& root1)
```

返回值：

无

函数功能：

以姓名为关键字，将目标结点插入到辅助结构（姓名树）中。

函数参数：

前者为目标结点地址，后者为二叉搜索树的根地址。

实现方式：

执行二叉搜索树的插入算法即可。

代码：

```
//返回值：无
//函数功能：以姓名为关键字，将目标结点插入到姓名树中
//函数参数：前者为目标结点地址，后者为姓名树的根地址
bool studentDataBase::insertByName(student* p, nameTreeNode*& root1)
{
    if (root1 == 0)
    {
        root1 = new nameTreeNode;
        strcpy_s(root1->name, p->name);
        root1->pointer = p;//使得辅助结点的 pointer 成员指向目标结点
        return true;
    }
    else if (strcmp(p->name, root1->name) < 0)return insertByName(p, root1->leftChild);
    else if (strcmp(p->name, root1->name) > 0)return insertByName(p, root1->rightChild);
    else return false;
}
```

insertByNum()

函数原型：

```
bool studentDataBase::insertByNum(student*p, numberTreeNode*& root1)
```

返回值：

无

函数功能：

以学号为关键字，将目标结点插入到辅助结构（学号树）中。

函数参数：

前者为目标结点地址，后者为二叉搜索树的根地址。

实现方式：

执行二叉搜索树的插入算法即可。

代码：

```
//返回值：无
//函数功能：以学号为关键字，将目标结点插入到学号树中
//函数参数：前者为目标结点地址，后者为学号树的根地址
bool studentDataBase::insertByNum(student*p, numberTreeNode*& root1)
{
    if (root1 == 0)
    {
        root1 = new numberTreeNode;
        strcpy_s(root1->number, p->number);
        root1->pointer = p;//使得辅助结点的 pointer 成员指向目标结点
        return true;
    }
    else if (strcmp(p->number, root1->number) < 0) return insertByNum(p, root1->leftChild);
    else if (strcmp(p->number, root1->number) > 0) return insertByNum(p, root1->rightChild);
    else return false;
}
```

insertByPhoneNum()

函数原型：

```
bool studentDataBase::insertByName(student* p, nameTreeNode*& root1)
```

返回值：

无

函数功能：

以电话为关键字，将目标结点插入到辅助结构（电话树）中。

函数参数：

前者为目标结点地址，后者为二叉搜索树的根地址。

实现方式：

执行二叉搜索树的插入算法即可。

代码：

```
//返回值：无
//函数功能：以姓名为关键字，将目标结点插入到姓名树中
```

```
//函数参数：前者为目标结点地址，后者为姓名树的根地址
bool studentDataBase::insertByName(student* p, nameTreeNode*& root1)
{
    if (root1 == 0)
    {
        root1 = new nameTreeNode;
        strcpy_s(root1->name, p->name);
        root1->pointer = p;//使得辅助结点的 pointer 成员指向目标结点
        return true;
    }
    else if (strcmp(p->name, root1->name) < 0)return insertByName(p, root1->leftChild);
    else if (strcmp(p->name, root1->name) > 0)return insertByName(p, root1->rightChild);
    else return false;
}
```

createTree()

函数原型：

```
void studentDataBase::createTree()//新建
```

返回值：

无

函数功能：

新建通讯录信息，用于依次输入信息并以#结束

实现方式：

采用层次序遍历的方式建立二叉树。因为用户输入的学生数据大一等于 1 条，所以可以先读入一条数据并作为根结点，将其入队后，执行 while 循环，声明 temp 为指向 student 的指针，赋值为队列顶端的元素，之后读取一次数据，先判断输入是否为#，若是#，就跳出 while 循环，若不是，就将输入作为 temp 的左孩子（同时也要将该结点插入三棵辅助树中），之后将新输入的结点入队。之后再读取一次数据，若不是#，就可以作为 temp 的右孩子，然后将指向该结点的指针压入队列。在此 while 循环的基础上，就可以完成以层次序遍历建立二叉树了。

代码：

```
//返回值：无
//函数功能：新建通讯录信息
```

```
void studentDataBase::createTree()
{
    queue<student*>stu;
    root = new student;
    cin >> *root;
    MIN = root;
    stu.push(root);//根结点入队
    while (!stu.empty())
    {
        student* temp = stu.front();//队首元素出队
        stu.pop();
        student* p = new student;//左子树
        cin >> *p;
        if (p->number[0]=='#')break;
        insertByName(p, nameTree);//在三棵辅助树中也插入
        insertByNum(p, numberTree);
        insertByPhoneNum(p, phoneNumTree);
        temp->leftChild= p;
        stu.push(p);
        p = new student;//右子树
        cin >> *p;
        if (p->number[0] == '#')break;
        insertByName(p, nameTree);//在三棵辅助树中也插入
        insertByNum(p, numberTree);
        insertByPhoneNum(p, phoneNumTree);
        temp->rightChild = p;
        stu.push(p);
    }
}
```

searchNum()

函数原型：

```
student* studentDataBase::searchNum(numberTreeNode*root,char num[])
```

返回值：

指向目标节点的指针

函数功能：

按照学号查找学生的通讯录信息。

函数参数：

前者是辅助结构（学号树）的根结点地址，后者是待查询的学号。

实现方式：

执行二叉搜索树的搜索算法即可，要注意的是，因为搜索树是一个辅助结构，里面只存有关键字和指向目标结点的地址。所以在搜索树中找到目标结点时，应该返回目标结点的 pointer 属性值，这样得到的才是指向二叉树中目标节点的地址。

代码：

```
//返回值：指向目标结点的指针
//函数功能：按学号查询信息
//函数参数：前者为 numberTree 的根节点，后者为待查询的学号
student* studentDataBase::searchNum(numberTreeNode*root,char num[])
{
    if (root == 0)return 0;
    else if (strcmp(root->number, num) > 0)return searchNum(root->leftChild, num);
    else if (strcmp(root->number, num) < 0)return searchNum(root->rightChild, num);
    else return root->pointer;
}
```

searchName()

函数原型：

```
student* studentDataBase::searchName(nameTreeNode*root,char name[])
```

返回值：

指向目标节点的指针

函数功能：

按照姓名查找学生的通讯录信息。

函数参数：

前者是辅助结构（姓名树）的根结点地址，后者是待查询的姓名。

实现方式：

执行二叉搜索树的搜索算法即可，要注意的是，因为搜索树是一个辅助结构，里面只存有关键字和指向目标结点的地址。所以在搜索树中找到目标结点时，应该返回目标结点的 pointer 属性值，这样得到的才是指向二叉树中目标节点的地址。

代码：

```
//返回值：指向目标结点的指针
//函数功能：按姓名查询信息
//函数参数：前者为 nameTree 的根节点，后者为待查询的姓名
student* studentDataBase::searchName(nameTreeNode*root,char name[])
{
    if (root == 0)return 0;
    else if (strcmp(root->name, name) > 0)return searchName(root->leftChild, name);
    else if (strcmp(root->name, name) < 0)return searchName(root->rightChild, name);
    else return root->pointer;
}
```

searchPhoneNum()

函数原型：

```
student* studentDataBase::searchPhoneNum(phoneNumTreeNode*root,char phoneNum[])
```

返回值：

指向目标节点的指针

函数功能：

按照电话查找学生的通讯录信息。

函数参数：

前者是辅助结构（电话树）的根结点地址，后者是待查询的电话。

实现方式：

执行二叉搜索树的搜索算法即可，要注意的是，因为搜索树是一个辅助结构，里面只存有关键字和指向目标结点的地址。所以在搜索树中找到目标结点时，应该返回目标结点的 pointer 属性值，这样得到的才是指向二叉树中目标节点的地址。

代码：

```
//返回值：指向目标结点的指针
//函数功能：按电话号码查询信息
//函数参数：前者为 phoneNumTree 的根节点，后者为待查询的电话号码
student* studentDataBase::searchPhoneNum(phoneNumTreeNode*root,char phoneNum[])
{
    if (root == 0)return 0;
```

```
    else if (strcmp(root->phoneNumber, phoneNum) > 0)return searchPhoneNum(root->leftChild, phoneNum);
    else if (strcmp(root->phoneNumber, phoneNum) < 0)return searchPhoneNum(root->rightChild, phoneNum);
    else return root->pointer;
}
```

delByNum()

函数原型：

```
bool studentDataBase::delByNum(char num[])//删除(搜索树按学号删除)
```

返回值：

逻辑值 0 或 1。

函数功能：

从二叉树中删除学号为 num 的学生信息。（实际上执行的是假删除，把目标节点的 isDeleted 值改成了 1）

函数参数：

待删除学号.

实现方式：

先调用搜索函数，得到目标节点的地址，再将目标结点的 isDeleted 值改成 1 即可。

代码：

```
//返回值：逻辑值 0 或 1
//函数功能：按照学号，将目标结点的 isDeleted 属性置为 1
//函数参数：待删除的学号
bool studentDataBase::delByNum(char num[])//删除(搜索树按学号删除)
{
    student* p = searchNum(numberTree, num);//此时 p 是目标结点
    if (p)
    {
        p->isDeleted = 1;
        cout << "删除成功！" << endl;
        return 1;
    }
```

```
    else
        cout << "无相关信息！" << endl;
    return 0;
}
```

delByName()

函数原型：

```
bool studentDataBase::delByName(char name[])//删除(搜索树按姓名删除)
```

返回值：

逻辑值 0 或 1。

函数功能：

从二叉树中删除姓名为 name 的学生信息。（实际上执行的是假删除，把目标节点的 isDeleted 值改成了 1）

函数参数：

待删除姓名.

实现方式：

先调用搜索函数，得到目标节点的地址，再将目标结点的 isDeleted 值改成 1 即可。

代码：

```
//返回值：逻辑值 0 或 1
//函数功能：按照姓名，将目标结点的 isDeleted 属性置为 1
//函数参数：待删除的姓名
bool studentDataBase::delByName(char name[])//删除(搜索树按姓名删除)
{
    student* p = searchName(nameTree, name);//此时 p 是目标结点
    if (p)
    {
        p->isDeleted = 1;
        cout << "删除成功！" << endl;
        return 1;
    }
    else
        cout << "无相关信息！" << endl;
```

```
    return 0;
}
```

delByPhoneNum()

函数原型：

```
bool studentDataBase::delByPhoneNum(char phoneNum[])//删除(搜索树按电话号码删除)
```

返回值：

逻辑值 0 或 1。

函数功能：

从二叉树中删除电话为 phoneNum 的学生信息。(实际上执行的是假删除，把目标节点的 isDeleted 值改成了 1)

函数参数：

待删除电话.

实现方式：

先调用搜索函数，得到目标节点的地址，再将目标结点的 isDeleted 值改成 1 即可。

代码：

```
//返回值：逻辑值 0 或 1
//函数功能：按照电话号码，将目标结点的 isDeleted 属性置为 1
//函数参数：待删除的电话号码
bool studentDataBase::delByPhoneNum(char phoneNum[])//删除(搜索树按电话号码删除)
{
    student* p = searchPhoneNum(phoneNumTree, phoneNum);//此时 p 是目标结点
    if (p)
    {
        p->isDeleted = 1;
        cout << "删除成功！" << endl;
        return 1;
    }
    else
        cout << "无相关信息！" << endl;
    return 0;
}
```

getYoungest()

函数原型：

```
void studentDataBase::getYoungest(student*root)
```

返回值：

无

函数功能：

使 MIN 指向年龄最小的学生。

函数参数：

二叉树的根结点地址。

实现方式：

采用层次序遍历的方式遍历二叉树，对于每一个结点，若该结点学生年龄小于 MIN 结点，并且该结点没有被删除（isDeleted 属性不为 1），则 MIN 可赋值为该结点地址.

代码：

```
//返回值：无
//函数功能：输出年龄最小的学生的信息
//函数参数：二叉树的根地址
void studentDataBase::getYoungest(student*root)
{
    //按照前序遍历的方式
    if (root == 0)return;
    else
    {
        //若该结点学生年龄小于 MIN 结点，并且该结点没有被删除，则 MIN 可赋值
        if (root->isDeleted==0&&strcmp(root->birthday,MIN->birthday) > 0)
            MIN = root;
        getYoungest(root->leftChild);
        getYoungest(root->rightChild);
    }
}
```

previsit()，midvisit()，behvisit()

函数原型：

```
void studentDataBase::previsit(student* root)//前序遍历输出
void studentDataBase::midvisit(student* root)//中序遍历输出
void studentDataBase::behvisit(student* root)//后序遍历输出
```

返回值：

无

函数功能：

按照前序、中序、后序遍历的方式输出学生通讯信息。

函数参数：

根结点地址

实现方式：

分别按照前序/中序/后序的遍历方式遍历每一个结点，对于每一个结点，如果 isDeleted 属性为 0，表示没有被删除，可以输出，若为 1，则表示已经被删除，不输出。

代码：

```
//返回值：无
//函数功能：前序输出信息
void studentDataBase::previsit(student* root)
{
    if (root == 0)return;
    else
    {
        if(!root->isDeleted)
            cout << *root << endl;
        previsit(root->leftChild);
        previsit(root->rightChild);
    }
}

//返回值：无
//函数功能：中序输出信息
```

```
void studentDataBase::midvisit(student* root)
{
    if (root == 0)return;
    else
    {
        midvisit(root->leftChild);
        if (!root->isDeleted)
            cout << *root << endl;
        midvisit(root->rightChild);
    }
}

//返回值：无
//函数功能：后序输出信息
void studentDataBase::behvisit(student* root)
{
    if (root == 0)return;
    else
    {
        behvisit(root->leftChild);
        behvisit(root->rightChild);
        if (!root->isDeleted)
            cout << *root << endl;
    }
}
```

output()

函数原型：

```
void studentDataBase::output()//输出所有信息
```

返回值：

无

函数功能：

根据用户的选择，分别调用 previsit(),midvisit(),behvisit()函数输出所有学生通讯录信息。

函数参数：

无

实现方式：

用 swtich 语句调用相应函数即可。对于每一个结点，如果 isDeleted 属性为 0，表示没有被删除，可以输出，若为 1，则表示已经被删除，不输出。

代码：

```
void studentDataBase::output()//输出所有信息
{
    cout << "你希望按照什么顺序输出？1(前序) 2(中序) 3(后序)" << endl;
    int input;
    cin >> input;
    if (input == 1)
    {
        title();
        previsit(root);
    }
    else if (input == 2)
    {
        title();
        midvisit(root);
    }
    else if (input == 3)
    {
        title();
        behvisit(root);
    }
    else
        cout << "请输入正确的指令！" << endl;
}
```

readFile()

函数原型：

```
void studentDataBase::readFile(string fileName)//读取文件
```

返回值：

无

函数功能：

将学生通讯信息从指定文件中读出以层次序遍历的方式建立二叉树，同时也建立起辅助结构（姓名树，学号树，电话树）。

函数参数：

要读取的文件名

实现方式：

打开二进制文件后，用 read 函数一次读取一个 student 结构体对象，并调用 insert()函数，insertByName(),insertByNum()，insertByPhoneNum()即可。读取完成后，将 MIN 初始化为 root。

代码：

```
//返回值：无
//函数功能：从目标文件读取信息到系统中
//函数参数：待读取文件名
void studentDataBase::readFile(string fileName)
{
    deleteTrees();
    ifstream infile(fileName, ios::binary);
    if (!infile)
    {
        cout << "open error!" << endl;
        return;
    }
    student* p;
    while (infile.peek() != EOF)
    {
        p = new student;
        infile.read((char*)&*p, sizeof(student));
        p->leftChild = 0;  //注意一定要初始化为 0，不然会乱指
        p->rightChild = 0;
        insert(p,root);//将该结点插入二叉树
        insertByName(p, nameTree);//同时插入三棵辅助树
        insertByNum(p, numberTree);
        insertByPhoneNum(p, phoneNumTree);
    }
    MIN = root;//将 MIN 初始化为 root
    infile.close();
}
```

writeFile()

函数原型：

```
void studentDataBase::writeFile(string fileName)//写入文件
```

返回值：

无

函数功能：

将系统中的信息写入到指定的文件中。

函数参数：

要读取的文件名

实现方式：

采用二进制形式打开文件后，用层次序遍历的方式将每个结点的信息都写到文件中。

代码：

```
void studentDataBase::writeFile(string fileName)//写入文件
{
    ofstream outfile(fileName, ios::binary);
    if (!outfile)
    {
        cout << "open error!" << endl;
        exit(1);
    }
    queue<student*>stu;
    student* p = root;
    stu.push(p);
    while (!stu.empty())
    {
        p = stu.front();
        stu.pop();
        if (p->leftChild)stu.push(p->leftChild);
        if (p->rightChild)stu.push(p->rightChild);
        outfile.write((char*)&(*p), sizeof(*p));
    }
    outfile.close();
```

```
}
```

randomData.cpp 实现

该文件包含仅包含一个 emergeData()函数,其功能是产生百万随机数据，并存储在 randomData.dat 文件中。

emergeData()

函数原型：

```
void emergeData()
```

返回值：

无

函数功能：

产生百万随机数据，并存储在 randomData.dat 文件中。

函数参数：

无

实现方式：

利用 rand()函数，产生随机数据，将数据转换成对应的类型后赋值给 student 结构体的对象，之后将该对象写入文件中。将此过程重复 100 万次即可生成百万随机数据。

代码：

```
//返回值：无
//函数功能：随机产生百万数据，并写入到randomData.dat文件中
void emergeData()
{
    student* p;
    ofstream outfile("randomData.dat", ios::binary);
    if (!outfile)
    {
        cout << "open error" << endl;
        abort();
    }
```

```
    cout << "正在生成，请稍后……" << endl;
    for (int i = 0; i <= 1000000; i++)
    {
        p = new student;
        sprintf_s(p->number, "%d",rand()%1000000);//把随机产生的数字转换成字符数组
        sprintf_s(p->name, "%d", rand() % 100000);
        sprintf_s(p->birthday, "%d", rand() % 10000000 + 19000000);
        sprintf_s(p->gender, "%d", rand() % 2);
        sprintf_s(p->phoneNumber, "%d", rand() % 1000000000 + 1000000000);
        sprintf_s(p->address, "%d", rand() % 10000);
        p->leftChild = 0;
        p->rightChild =0;
        outfile.write((char*)&(*p), sizeof(student));
        delete p;
    }
    outfile.close();
    cout << "生成成功！" << endl;
}
```

main.cpp 实现

为实现更快速度的交互过程，采用了函数指针数组来实现相应函数的调用。在函数 func1-func12 中，分别调用新建、插入、按学号查询、按姓名查询、按电话号码查询、删除、输出年龄最小的信息、输出所有信息、读取、写入、读入百万数据、退出函数。函数指针数组声明如下：

```
//函数指针数组的内容
//利用函数指针数组，避免使用了switch结构来实现相应函数的调用，提升效率
void func1(void);//第1到11个功能调用的函数
void func2(void);
void func3(void);
void func4(void);
void func5(void);
void func6(void);
void func7(void);
void func8(void);
void func9(void);
void func10(void);
void func11(void);
void func12(void);

//函数指针数组，用于根据用户输入调用对应函数
```

```
void(*functionPointer[12])(void) =
{ func1,func2,func3,func4,func5,func6,func7,func8,func9,func10,func11,func12 };
```

之后，在 main 函数中，根据用户输入，即可完成相应函数的调用，代码如下：

(说明：menu()函数实现的是显示菜单的功能，此处不做介绍)

```
//主函数
//函数功能：实现用户交互，调用对应函数
int main()
{
    int input;
    menu();
    while (true)
    {
        cin >> input;
        if (input >= 1 && input <= 12)
        {
            (*functionPointer[input - 1])();//调用对应函数
            menu();//显示菜单
        }
        else
            cout << "请输入正确的指令！" << endl;
    }
   //emergeData();
    return 0;
}
```

至此，实现部分介绍完成。

- 百万数据已经生成并存储在 randomData.dat 文件中，可以在系统中选择相应功能实现读取和管理。
- 当学生数据超过千万的时候，系统效率极大降低，按照系统管理百万数据的耗时来推测，当学生数据超过千万的时候，读取数据需要耗时 1 到 2 小时，执行查找、插入、删除的操作可能会耗时几十秒。
- 接下来是研究分析报告书。

研究分析报告书

对 100 万条数据进行操作时，从文件读数据过程耗时很长，达到了 10 分钟左右（可能时同时生成了太多辅助结构的原因）。但是对数据操作的速度不是很慢，执行插入、按学号、姓名、电话号码查询、删除时，耗时 3-4 秒，但比起小规模的数据，这个耗时可以说很长了。可以说，当数据到达百万时，整个系统的执行速度变得很慢，针对这个问题，以下分析原因、以及提出解决方法。

效率分析：

本次实验中实现查找、删除时，用到了辅助结构二叉搜索树，总的时间复杂度为 O(log2(n))，速度相对较快。导致系统读取速度变慢的原因应该是在读取的同时建立了四棵树（一棵二叉树和三棵用于辅助的树），导致计算量加大，效率变慢。

在输出年龄最小的学生时，用的时层次序遍历的方式，复杂度为 O(n)，当数据达到百万时，效率会大大降低。

算法改进:

① 在读取文件时，考虑只按照层次序遍历生成一棵二叉树。其他三棵用于辅助的树可以等到使用到相应功能的时候在生成，这样可以提高读取速度。

② 如果比较常查找某个关键字，也可以建一个哈希表，如果是以姓名为关键字可以使用字符串哈希 KMP 等算法。这样复杂度会降为 O(1);

③ 可以建立以年龄为关键字的小根堆来实现查找年龄最小的学生，这样时间复杂度可以直接降到 O(1)。

四、实验总结（心得体会）

① 要注意实验时间的规划，若是堆在几天完成，容易手忙脚乱，实现思路不清晰。

② 缺少图形化编程的基础，我完成的利用 Qt 实现的图形化程序的组织架构很乱，不像控制台实现方式那样有条理。后续需要进一步优化组织架构。

③ 吸收实验一的经验，考虑到用户隐私和数据安全，本次文件读取不再使用 ascll 文件，而是采用二进制文件读写方式。

④ 我发现生成百万数据后，系统在读取数据时速度明显变慢，而且在读取到几十万条数据的时候，就会自动结束读取文件的函数（实际测试时读取了三万多条），我暂时没有找到引发这个现象的原因，推测是 visual studio 平台设置的原因，后续将对这一现象进行处理。(2021 年 1 月 1 日，现在已经解决该问题，具体实现方法是把 string 换成字符数组)

⑤ 重新审视我写的代码，发现有些能够复用代码的地方我没有做到复用，导致代码较为复杂，此外，对于各个函数的代码实现也不够精简。究其原因，我认为是我在写代码之前，没有做好系统的具体规划，而是想到什么就写什么。下次注意。

五、参考文献：

1、 《数据结构：用面向对象方法和 C++语言描述》殷人昆主编

2、 由二叉树转换成二叉排序树

网址：https://blog.csdn.net/u012609067/article/details/31797061