

数据结构大型实验

**2018/2019(1)**



实验题目：用户登录系统的模拟

学生姓名：高源

学生学号：201706062013

学生班级：软件工程1704

任课教师：王松

完成日期：2018年12月28日星期五

**计算机科学与技术学院**

**用户登录系统模拟实验报告**

# 实验题目和要求

**【问题描述】**在登录服务器系统时，都需要验证用户名和密码，如telnet远程登录服务器。用户输入用户名和密码后，服务器程序会首先验证用户信息的合法性。由于用户信息的验证频率很高，系统有必要有效地组织这些用户信息，从而快速查找和验证用户。另外，系统也会经常会添加新用户、删除老用户和更新用户密码等操作，因此，系统必须采用动态结构，在添加、删除或更新后，依然能保证验证过程的快速。请采用相应的数据结构模拟用户登录系统，其功能要求包括用户登录、用户密码更新、用户添加和用户删除等。

**【基本要求】**

1. 要求自己编程实现二叉树结构及其相关功能，以存储用户信息，**不允许使用标准模板类的二叉树结构和函数**。同时要求根据二叉树的变化情况，进行相应的平衡操作，即AVL平衡树操作，**四种平衡操作都必须考虑**。测试时，各种情况都需要测试，并附上测试截图；
2. 要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。主函数中只能出现类的成员函数的调用，不允许出现对其它函数的调用。
3. 要求采用多文件方式：.h文件存储类的声明，.cpp文件存储类的实现，主函数main存储在另外一个单独的cpp文件中。如果采用类模板，则类的声明和实现都放在.h文件中。
4. 不强制要求采用类模板，也不要求采用可视化窗口；要求源程序中有相应注释；
5. 要求测试例子要比较详尽，各种极限情况也要考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确；
6. 要求采用Visual C++ 6.0及以上版本进行调试；

# 设计思路

## 系统总体设计

运用AVL树，以用户的账号和密码作为信息建立节点，实现AVL树的插入，删除，登录验证，从文件读取信息建立AVL树，与程序运行结束后将树中的信息写回文件，并同时保持树的高度平衡。

## 系统功能设计

|  |  |  |
| --- | --- | --- |
| 登录选项 | 选项 | 说明 |
| 用户登录 | 修改密码 | 先输入旧密码，再输入新密码，如果旧密码正确，修改密码成功，否则失败 |
| 永久注销该用户 | 从数据文件中删除正在登录的用户并返回至初始登录页面 |
| 返回上一级 | 返回初始登录页面 |
| 管理员登录 | 修改密码 | 先输入旧密码，再输入新密码，如果旧密码正确，修改密码成功，否则失败 |
| 注册新的管理员账户 | 输入新账户与新密码注册管理员账号 |
| 删除用户 | 输入需要删除的用户账户来进行删除，若存在则删除，不存在则返回删除失败 |
| 增加用户 | 增加新的用户账户 |
| 返回上一级 | 返回初始登录界面 |
| 注册用户 | 输入新账户与密码注册新的用户，若已经存在则返回创建失败，已有同名用户。反之创建成功 | |
| 退出 | 退出系统，并将AVL树中的信息写入文件 | |

## 类的设计

### 全部类的成员变量与函数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user | | | | |
| 权限 | 分类 | 类型||返回类型 | 名称 | 说明 |
| public | 成员变量 | string | account | 账户 |
| string | password | 密码 |
| int | height | 以该节点为根的树的高度 |
| user \* | left | 左子树 |
| user \* | right | 右子树 |
| 成员函数 | 构造函数 | user(string ac,string pa) | 用ac,pa初始化一个节点，height置为0，其他项置为空 |
| 构造函数 | user() | 初始化 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| userAVLtree | | | | |
| 权限 | 分类 | 类型 | 名称 | 说明 |
| public | 成员函数 | 构造函数 | userAVLtree() | 成员变量root置为null |
| 析构函数 | ~userAVLtree（） | 利用destroy()析构整棵AVL树 |
| int | height() | 获取整棵树的高度 |
| int | max(int a,int b) | 返回a,b之中最大的树 |
| void | readfile(ifstream & infile) | 将用户信息从文件写入到树里 |
| void | writetofile(ofstream & outfile) | 将avl树中的信息写到文件里 |
| void | insert(string ac,string pa) | 插入新的用户至树里 |
| void | deleteuser(string ac) | 删除用户 |
| bool | loginverify(string ac,string pa) | 用户登录验证 |
| void | updatepassword(string ac,string oldpassword,string newpassword) | 用户更新密码 |
| user \* | search(string ac) | 查找函数返回目标的user \*，用来辅助登录验证和更新密码 |
| void | destroy() | 摧毁整棵树 |
| private | user \* | maximum(user \*subtree) | 查找以subtree为根的树下的最小节点，返回user \* |
| user \* | minimum(user \*subtree) | 查找以subtree为根的树的最小节点，返回user \* |
| void | height(user \*subtree) | 获取以该节点为根的树的高度 |
| user \* | insertAux(string ac,string pa,user \*subtree) | 插入函数辅助函数 |
| user \* | deleteuserAux(user \*target,user \*subtree) | 删除节点函数辅助函数 |
|  | leftleftRotation(user \*k2) | 右旋，要插入的节点位于k2左孩子的左子树 |
|  | rightrightRotation(user \*k2) | 左旋，要插入节点位于k2右孩子的右子树 |
|  | leftrightRotation(user \*k1) | 左-右旋 |
|  | rightleftRotation(user \*k1) | 右-左旋 |
|  | destroy(user \* tree) | 辅助删除树的全部节点 |
|  | writetofileAux(ofstream & outfile,user \* tree) | 辅助将树的节点数据输入到文件中 |
| 成员变量 | user \* | root |  |

### 类之间的关系

userAVLtree

成员变量

user

### 对关键函数的阐述

#### user \*insertAux(string ac,string pa,user \*subtree)

开始

subtree=new user(ac,pa)

subtree->left=insertAux(ac,pa,subtree->left)

Subtreee==null user(ac,pa)

ac<subtree->account

ac>subtree->account

如果在左孩子的左子树中，右旋，反之左-右旋

subtree->right=insertAux(ac,pa,subtree->right)

输出添加失败，相同节点

如果在右孩子的右子树中左旋，反之右-左旋

修改subtree的高度为左右子树最大高度加1

return subtree

结束

Y

Y

Y

N

N

N

### user \*deleteuserAux(user \*target,user \*subtree);//target为目标节点

开始

target->account<subtree->account

subtree->left=deleteuserAux(target,subtree->left);

大于

递归调用subtree->right

不平衡

右孩子左子树高度

大于右子树高度

右旋

左右旋

不平衡

左孩子右子树高度

大于左子树高度

左旋

右-左旋

左右子女皆不为空

1. 见下

用非空

子女代替该节点

Return subtree

结束

//如果左子树高

//将subtree左子树最大节点的账号密码赋给subtree

//删除该最大节点

if(height(subtree->left)>height(subtree->right))

{

user \* max1=maximum(subtree->left);

subtree->account=max1->account;

subtree->password=max1->password;

subtree->left=deleteuserAux(max1,subtree->left);

}

else

{

//如果右子树高或两边高度相等

//将subtree右子树最小节点的账号密码赋给subtree

//删除该最大节点

user \* min1=minimum(subtree->right);

subtree->account=min1->account;

subtree->password=min1->password;

subtree->right=deleteuserAux(min1,subtree->right);

}

## 主程序设计

将用户的数据保存至userdata.txt中，管理员的数据保存于manager.txt中，程序开始时从这两个文件中读取数据建立树users和managers

以循环和嵌套的switch-case语句来实现菜单，类似

for(;;)

{

//输出菜单

cout<<4.退出 <<end;………………..

cout<<endl;

cin>>i;

if(i==4)

break;

switch(i)

{

case 1:

{

//语句

break;

}

case 2:

{

//语句

break;

}

.....

default:

}

}

在语句中调用users和managers的成员函数实现各种功能

程序结束后将users和managers树中的数据写入文件

# 调试分析

## 技术难点分析

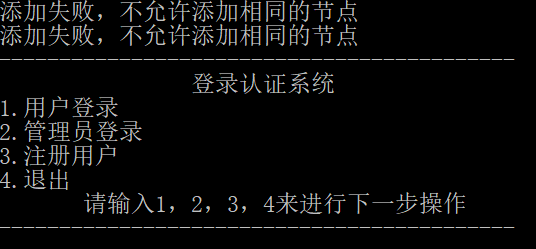
最大的技术难点是插入和删除之后如何保持树的平衡性

以插入为例

插入之后，需要一级一级向上确认树的平衡性，如果不平衡，判断出插入的节点位于subtree的什么位置（如左孩子右子树等等）从而进行相应的旋转操作，保持树的平衡性。

## 调试错误分析

程序完成之后，运行显示文件最后一个账户密码插入了两次树，如下



经过，发现了如下问题代码

void userAVLtree::readfile(ifstream &infile)

{

string ac;

string pa;

while(!infile.eof())

{

infile>>ac>>pa;

insert(ac,pa);

}

}

其中当infile读取完最后一行数据后，只有接着向下读取才能使得infile.eof()变为真，所以重新又读取，事实上此时是读取失败的

修改后的代码如下

void userAVLtree::readfile(ifstream &infile)

{

string ac;

string pa;

while(!infile.eof())

{

infile>>ac>>pa;

//防止最后一行数据读取两次

if(!infile)

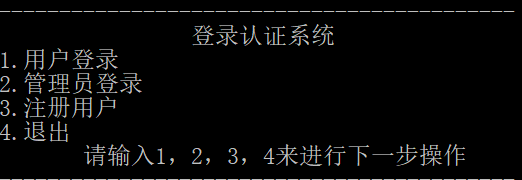
break;

insert(ac,pa);

}

}

运行截图如下



问题代码如下

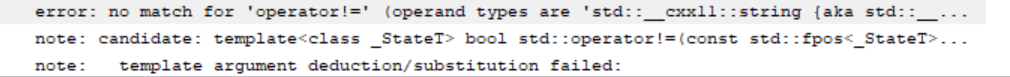
if(tree->account!=0&&tree->password!=0)

{

outfile<<tree->account<<" "<<tree->password<<endl;

}

编译错误截图



这段代码的意思是如果string类的account和password不为空，则输出至文件，但编译之后系统提示一大段错误

在网上查询之后得知，string是无法被初始化成NULL的，所以只能用string.length()==0或string.empty()来判断是否为空

改善后的代码如下

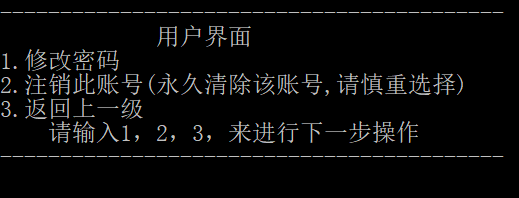
if(tree->account.length()!=0&&tree->password.length()!=0)

{

outfile<<tree->account<<" "<<tree->password<<endl;

}

因为考虑欠周，导致在用户界面注销用户之后返回的仍是用户界面而不是初始登录界面



后修改代码如下

else if(second==2)

{

//注销用户

system("cls");

users.deleteuser(account);

system("pause");

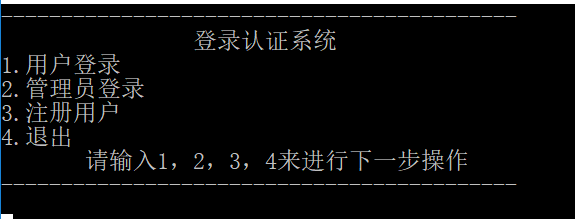
break;

}

可以正常返回初始登录界面

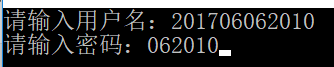
# 测试结果分析

## 初始界面



## 输入1

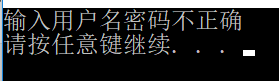
用户登录界面



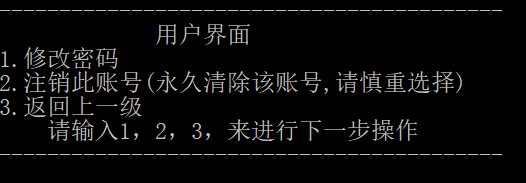
输入正确



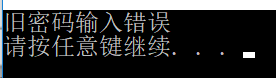
输入错误



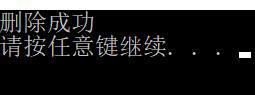
用户界面



用户修改密码，旧密码输入错误



永久注销用户

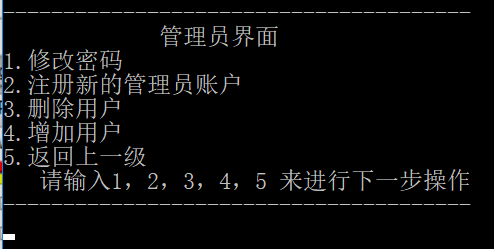


返回上一级——返回登录页面

## 输入2：

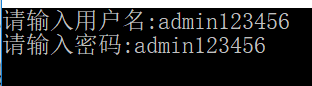
管理员登录，登录界面同上

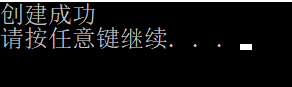
管理员界面



修改密码同上

注册新的管理员账户





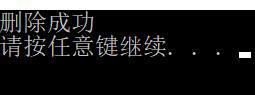
删除用户



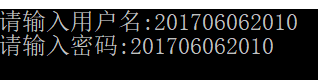
若用户不存在



用户存在



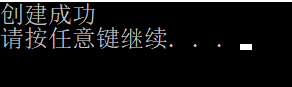
增加用户



若用户已存在



若用户不存在



返回上一级——返回初始页面

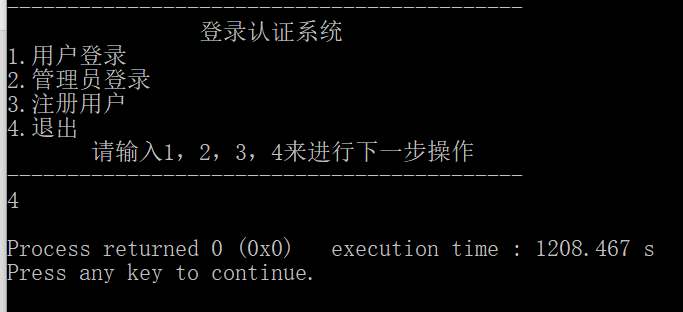
## 输入3

注册用户

同管理员选项中的新建用户

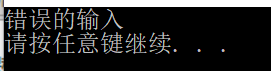
## 输入4

退出系统



## 若输入非1,2,3,4

则显示



之后返回原来的界面

# 源代码附录

## userAVLtree.h

#ifndef USERAVLTREE\_H

#define USERAVLTREE\_H

#include<iostream>

#include<fstream>

using namespace std;

class user

{

public:

string account;

string password;

//树的高度

int height;

user \*left;

user \*right;

user(string ac,string pa):account(ac),password(pa),height(0),left(0),right(0){}

user():account(0),password(0),height(0),left(0),right(0){}

};

class userAVLtree

{

public:

userAVLtree();

virtual ~userAVLtree();

//获取树的总高度

int height();

int max(int a,int b);

//将用户信息从文件写入到树里

void readfile(ifstream & infile);

//将avl树中的信息写到文件里

void writetofile(ofstream & outfile);

//插入新的用户至树里

void insert(string ac,string pa);

//删除用户

void deleteuser(string ac);

//用户登录验证

bool loginverify(string ac,string pa);

//用户更新密码；

void updatepassword(string ac,string oldpassword,string newpassword);

//查找函数返回目标的user \*，用来辅助登录验证和更新密码

user \*search(string ac);

//摧毁树

void destroy();

private:

//查找以subtree为根的树下的最小节点，返回user \*

user \* maximum(user \*subtree);

//查找以subtree为根的树的最小节点，返回user \*

user \* minimum(user \*subtree);

//获取以该节点为根的树的高度

int height(user \*subtree);

//插入函数辅助函数

user \*insertAux(string ac,string pa,user \*subtree);

//删除节点函数辅助函数

user \*deleteuserAux(user \*target,user \*subtree);

//要插入的节点位于k2左孩子的左子树

user \*leftleftRotation(user \*k2);

//要插入节点位于k2右孩子的右子树

user \*rightrightRotation(user \*k2);

//位于左孩子的右子树

user \*leftrightRotation(user \*k1);

//位于右孩子的左子树

user \*rightleftRotation(user \*k1);

//辅助删除树的全部节点

void destroy(user \* tree);

//辅助将树的节点数据输入到文件中

void writetofileAux(ofstream & outfile,user \* tree);

user \* root;

};

#endif // USERAVLTREE\_H

## userAVLtree.cpp

#include "userAVLtree.h"

userAVLtree::userAVLtree()

{

root=0;

}

userAVLtree::~userAVLtree()

{

destroy();

}

/\*

\*求树的的高度

\*/

int userAVLtree::height()

{

return height(root);

}

int userAVLtree::height(user \* subtree)

{

if(subtree==0)

{

return 0;

}

return subtree->height;

}

/\*

\*比较两数大小

\*/

int userAVLtree::max(int a,int b)

{

return (a>b?a:b);

}

/\*

\*右旋

\*返回旋转后的根节点

\*/

user \* userAVLtree::leftleftRotation(user \* k2)

{

user \*k1=k2->left;

k2->left=k1->right;

k1->right=k2;

k2->height=max(height(k2->right),height(k2->left))+1;

k1->height=max(height(k1->right),height(k1->left))+1;

return k1;

}

/\*

\*左旋

\*返回旋转后的根节点

\*/

user \* userAVLtree::rightrightRotation(user \*k2)

{

user \*k1=k2->right;

k2->right=k1->left;

k1->left=k2;

k2->height=max(height(k2->left),height(k2->right))+1;

k1->height=max(height(k1->right),height(k1->left))+1;

return k1;

}

/\*

\*左-右旋

\*返回旋转后的根节点

\*/

user \*userAVLtree::leftrightRotation(user\* k1)

{

k1->left=rightrightRotation(k1->left);

return leftleftRotation(k1);

}

/\*

\*右-左旋

\*返回旋转后的根节点

\*/

user \*userAVLtree::rightleftRotation(user \* k1)

{

k1->right=leftleftRotation(k1->right);

return rightrightRotation(k1);

}

void userAVLtree::insert(string ac,string pa)

{

root=insertAux(ac,pa,root);

}

/\*

\*插入一个用户节点

\*返回根节点

\*/

user \*userAVLtree::insertAux(string ac,string pa,user\* subtree)

{

if(subtree==0)

{

subtree=new user(ac,pa);

if(subtree==0)

{

cout<<"creat node failed"<<endl;

return 0;

}

}

else if(ac<subtree->account)

{

subtree->left=insertAux(ac,pa,subtree->left);

if(height(subtree->left)-height(subtree->right)==2)

{

if(ac<subtree->left->account)

subtree=leftleftRotation(subtree);

else

subtree=leftrightRotation(subtree);

}

}

else if(ac>subtree->account)

{

subtree->right=insertAux(ac,pa,subtree->right);

if(height(subtree->right)-height(subtree->left)==2)

{

if(ac>subtree->right->account)

subtree=rightrightRotation(subtree);

else

subtree=rightleftRotation(subtree);

}

}

else//account=subtree->account

{

cout<<"添加失败，不允许添加相同的节点"<<endl;

}

subtree->height=max(height(subtree->left),height(subtree->right))+1;

return subtree;

}

/\*

\*按account查找节点

\*返回目标节点的user \*

\*/

user \* userAVLtree::search(string ac)

{

user\* locptr=root;

while(locptr!=0)

{

if(ac<locptr->account)

locptr=locptr->left;

else if(ac>locptr->account)

locptr=locptr->right;

else

return locptr;

}

return 0;

}

/\*

\*运用search函数验证用户名密码

\*返回正确与否

\*/

bool userAVLtree::loginverify(string ac,string pa)

{

user \* target;

target=search(ac);

if(target==0)

return false;

else if(target->password==pa)

return true;

else

return false;

}

/\*

\*前置:输入用户名，旧密码，新密码

\*后置:若旧密码正确，则更新，反之不更新

\*/

void userAVLtree::updatepassword(string ac,string oldpassword,string newpassword)

{

if(loginverify(ac,oldpassword))

{

user \*target=search(ac);

target->password=newpassword;

cout<<"更新成功"<<endl;

}

else

cout<<"旧密码输入错误"<<endl;

}

/\*

\*前置:输入节点指针

\*后置:返回以该指针为根的树的最大节点

\*/

user \* userAVLtree::maximum(user \*subtree)

{

if(subtree!=0)

{

while(subtree->right!=0)

{

subtree=subtree->right;

}

return subtree;

}

else

return 0;

}

/\*

\*前置:输入节点指针

\*后置:返回以该指针为根的树的最大节点

\*/

user \* userAVLtree::minimum(user \*subtree)

{

if(subtree!=0)

{

while(subtree->left!=0)

{

subtree=subtree->left;

}

return subtree;

}

else

return 0;

}

void userAVLtree::deleteuser(string ac)

{

user \*target;

if((target=search(ac))!=0)

{

root=deleteuserAux(target,root);

cout<<"删除成功"<<endl;

}

else

cout<<"用户不存在"<<endl;

}

/\*

\*前置:传入待删除节点的user \*和根

\*后置:删除节点，返回AVL树的根

\*/

user \* userAVLtree::deleteuserAux(user \*target,user \*subtree)

{

if(target->account<subtree->account)//待删除节点在subtree左子树中

{

subtree->left=deleteuserAux(target,subtree->left);

//删除节点后，若AVL树失去平衡则进行相应调节

if(height(subtree->right)-height(subtree->left)==2)

{

user \* temp=subtree->right;

if(height(temp->right)>height(temp->left))

subtree=rightrightRotation(subtree);

else

subtree=rightleftRotation(subtree);

}

}

else if(target->account>subtree->account)//待删除节点在subtree右子树中

{

subtree->right=deleteuserAux(target,subtree->right);

if(height(subtree->left)-height(subtree->right)==2)

{

user \* temp2=subtree->left;

if(height(temp2->left)>height(temp2->right))

subtree=leftleftRotation(subtree);

else

subtree=leftrightRotation(subtree);

}

}

else//待删除节点为subtree

{

//subtree子女节点皆不为空

if(subtree->left!=0&&subtree->right!=0)

{

//如果左子树高

//将subtree左子树最大节点的账号密码赋给subtree

//删除该最大节点

if(height(subtree->left)>height(subtree->right))

{

user \* max1=maximum(subtree->left);

subtree->account=max1->account;

subtree->password=max1->password;

subtree->left=deleteuserAux(max1,subtree->left);

}

else

{

//如果右子树高或两边高度相等

//将subtree右子树最小节点的账号密码赋给subtree

//删除该最大节点

user \* min1=minimum(subtree->right);

subtree->account=min1->account;

subtree->password=min1->password;

subtree->right=deleteuserAux(min1,subtree->right);

}

}

else

{

user \* temp2=subtree;

subtree=(subtree->left==0)?subtree->right:subtree->left;

delete temp2;

}

}

return subtree;

}

/\*

\*删除树的全部节点

\*/

void userAVLtree::destroy()

{

destroy(root);

}

void userAVLtree::destroy(user \* tree)

{

if (tree==NULL)

return ;

if (tree->left != NULL)

destroy(tree->left);

if (tree->right != NULL)

destroy(tree->right);

delete tree;

}

/\*

\*运用ifstream从指定文件一行一行读取用户名和密码

\*形成一个AVL用户树

\*/

void userAVLtree::readfile(ifstream &infile)

{

string ac;

string pa;

while(!infile.eof())

{

infile>>ac>>pa;

//防止最后一行数据读取两次

if(!infile)

break;

insert(ac,pa);

}

}

/\*

\*将树中的全部节点数据写入到文件中

\*/

void userAVLtree::writetofile(ofstream &outfile)

{

writetofileAux(outfile,root);

}

void userAVLtree::writetofileAux(ofstream &outfile,user \*tree)

{

if(tree!=0)

{

//前序遍历输出

if(tree->account.length()!=0&&tree->password.length()!=0)

{

outfile<<tree->account<<" "<<tree->password<<endl;

}

writetofileAux(outfile,tree->left);

writetofileAux(outfile,tree->right);

}

}

## main.cpp

#include <iostream>

#include"userAVLtree.h"

using namespace std;

int main()

{

//建立用户的AVL树

ifstream infileuser("userdata.txt",ios::in);

if(!infileuser)

{

cout<<"open user's txt failed"<<endl;

exit(1);

}

userAVLtree users;

users.readfile(infileuser);

infileuser.close();

//建立管理员的AVL树

ifstream infilemanager("manager.txt",ios::in);

if(!infilemanager)

{

cout<<"open manager's txt failed"<<endl;

exit(1);

}

userAVLtree managers;

managers.readfile(infilemanager);

infilemanager.close();

//用来保存用户输入的账号和密码

string account;

string password;

//记录用户进行的操作

int first,second,third,fourth;

for(;;)

{

//初始界面

cout<<"-------------------------------------------"<<endl;

cout<<" 登录认证系统 "<<endl;

cout<<"1.用户登录 "<<endl;

cout<<"2.管理员登录 "<<endl;

cout<<"3.注册用户 "<<endl;

cout<<"4.退出 "<<endl;

cout<<" 请输入1，2，3，4来进行下一步操作 "<<endl;

cout<<"-------------------------------------------"<<endl;

cin>>first;

//如果输入为4则退出系统

if(first==4)

break;

switch(first)

{

case 1:

{

//用户登录

system("cls");

cout<<"请输入用户名：";

cin>>account;

cout<<"请输入密码：";

cin>>password;

//如果用户名密码正确

if(users.loginverify(account,password))

{

system("cls");

cout<<"登录成功"<<endl;

system("pause");

for(;;)

{

system("cls");

//用户界面

cout<<"------------------------------------------"<<endl;

cout<<" 用户界面 "<<endl;

cout<<"1.修改密码 "<<endl;

cout<<"2.注销此账号(永久清除该账号,请慎重选择) "<<endl;

cout<<"3.返回上一级 "<<endl;

cout<<" 请输入1，2，3，来进行下一步操作 "<<endl;

cout<<"------------------------------------------"<<endl;

cin>>second;

if(second==3)

break;

else if(second==2)

{

//注销用户

system("cls");

users.deleteuser(account);

system("pause");

break;

}

switch(second)

{

case 1:

{

//改密码

system("cls");

string newpassword;

cout<<"请输入旧密码:";

cin>>password;

cout<<"请输入新密码:";

cin>>newpassword;

system("cls");

users.updatepassword(account,password,newpassword);

system("pause");

break;

}

default:

{

system("cls");

cout<<"输入错误,将跳回上级页面";

system("pause");

}

}

}

}

else//输入用户名密码不正确；

{

system("cls");

cout<<"输入用户名密码不正确"<<endl;

system("pause");

}

break;

}

case 2:

{

//管理员登录

system("cls");

cout<<"请输入用户名：";

cin>>account;

cout<<"请输入密码：";

cin>>password;

if(managers.loginverify(account,password))

{

system("cls");

cout<<"登录成功"<<endl;

system("pause");

for(;;)

{

system("cls");

//管理员界面

cout<<"---------------------------------------"<<endl;

cout<<" 管理员界面 "<<endl;

cout<<"1.修改密码 "<<endl;

cout<<"2.注册新的管理员账户 "<<endl;

cout<<"3.删除用户 "<<endl;

cout<<"4.增加用户 "<<endl;

cout<<"5.返回上一级 "<<endl;

cout<<" 请输入1，2，3，4，5 来进行下一步操作"<<endl;

cout<<"---------------------------------------"<<endl;

cin>>third;

if(third==5)

break;

switch(third)

{

case 1:

{

//改密码

system("cls");

string newpassword;

cout<<"请输入旧密码:";

cin>>password;

cout<<"请输入新密码:";

cin>>newpassword;

system("cls");

users.updatepassword(account,password,newpassword);

system("pause");

break;

}

case 2:

{

system("cls");

string tempaccount;

string temppassword;

cout<<"请输入用户名:";

cin>>tempaccount;

cout<<"请输入密码:";

cin>>temppassword;

managers.insert(tempaccount,temppassword);

system("cls");

cout<<"创建成功"<<endl;

system("pause");

break;

}

case 3:

{

system("cls");

string targetaccount;

cout<<"请输入要删除用户的用户名:";

cin>>targetaccount;

system("cls");

users.deleteuser(targetaccount);

system("pause");

break;

}

case 4:

{

system("cls");

string tempaccount1;

string temppassword1;

cout<<"请输入用户名:";

cin>>tempaccount1;

cout<<"请输入密码:";

cin>>temppassword1;

if(users.search(tempaccount1)!=0)

{

system("cls");

cout<<"创建失败，用户名相同"<<endl;

}

else

{

users.insert(tempaccount1,temppassword1);

system("cls");

cout<<"创建成功"<<endl;

}

system("pause");

break;

}

default:

{

system("cls");

cout<<"输入错误,将跳回上级页面";

system("pause");

}

}

}

}

else//用户名密码不正确

{

system("cls");

cout<<"输入用户名密码不正确"<<endl;

system("pause");

}

break;

}

case 3:

{

system("cls");

string tempaccount1;

string temppassword1;

cout<<"请输入用户名:";

cin>>tempaccount1;

cout<<"请输入密码:";

cin>>temppassword1;

if(users.search(tempaccount1)!=0)

{

system("cls");

cout<<"创建失败，用户名相同"<<endl;

}

else

{

users.insert(tempaccount1,temppassword1);

system("cls");

cout<<"创建成功"<<endl;

}

system("pause");

break;

}

default:

system("cls");

cout<<"错误的输入"<<endl;

system("pause");

}

system("cls");

}

ofstream outfileuser("userdata.txt",ios::out);

if(!outfileuser)

{

cout<<"write user data failed"<<endl;

exit(1);

}

users.writetofile(outfileuser);

outfileuser.close();

ofstream outfilemanager("manager.txt",ios::out);

if(!outfilemanager)

{

cout<<"write manager data failed"<<endl;

exit(1);

}

managers.writetofile(outfilemanager);

outfilemanager.close();

}