

《程序设计基础》大作业

——通讯录管理程序

Shiyuu

2023 年 4 月 21 日

1 项目分析

1.1 问题描述

通讯录管理系统

问题描述及设计要求：

设计一个通讯录管理系统，要求实现以下功能：

- (1) 添加联系人信息（包括电话号码、姓名、工作单位和住址），并可连续添加；
- (2) 查找联系人信息（按电话号码查找，返回联系人的所有信息）；
- (3) 删除联系人信息；
- (4) 修改联系人信息；
- (5) 对联系人信息按姓名首字母进行排序（升序）；
- (6) 统计联系人个数；
- (7) 显示所有联系人信息；
- (8) 退出系统。

附加功能：

- (1) 清空通讯录；
- (2) 检查输入的合法性（如电话号码必须为 11 位纯数字，不得出现字母、标点或中文）；
- (3) 在指定联系人信息记录后面插入新记录；
- (4) 以外部文件读写的方式保存通讯录。

1.2 问题分析

这是一个典型的数据管理型的程序。注意到题目要求的添加、查找、删除、修改等要求，考虑使用二叉树数据结构存储通讯录信息。为了保证二叉搜索树在数据量很大时的效率，考虑使用 AVL 二叉树数据结构。同时，题目要求我们将通讯录数据保存到外部文件中，所以还要编写相应的文件读写程序。为了程序的运行稳定性和可移植性，需要编写动态存储的程序。最后为了方便管理和修改功能，把项目中关于二叉树数据类型抽象为一个接口，封装到单独的文件中，也就是创建抽象数据类型（ADT），而管理交互界面（GUI）的主文件则单独列出，编译时进行多文件编译。

通过这次大作业，可以：

1. 掌握 AVL 二叉树的创建与操作
2. 掌握抽象数据结构 (ADT) 的创建与使用

3. 掌握多文件项目的基本方法
4. 进一步体会结构体的使用方法
5. 练习文件读写操作

1.3 基本知识

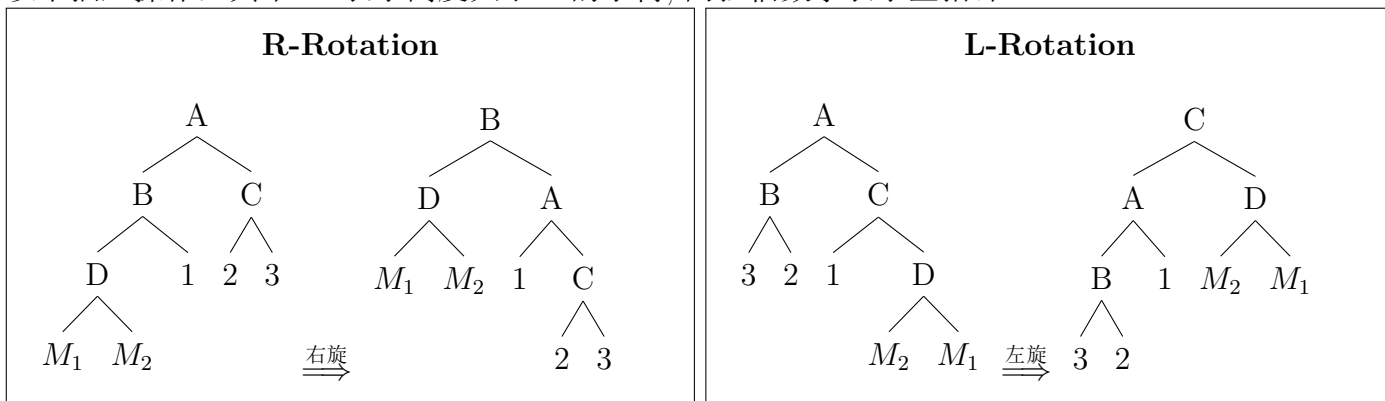
1.3.1 AVL 二叉树

AVL 树是一种自平衡二叉搜索树，最早由俄国数学家 Adel'son-Vel'skii 和 Landis 发明。它通过在树的结构体中植入一个高度因子，计算每个节点的左子树和右子树高度之差，通过左旋和右旋操作使这个差值不超过 1。

AVL 树的每个节点都存储了一个键值，并且满足以下性质：

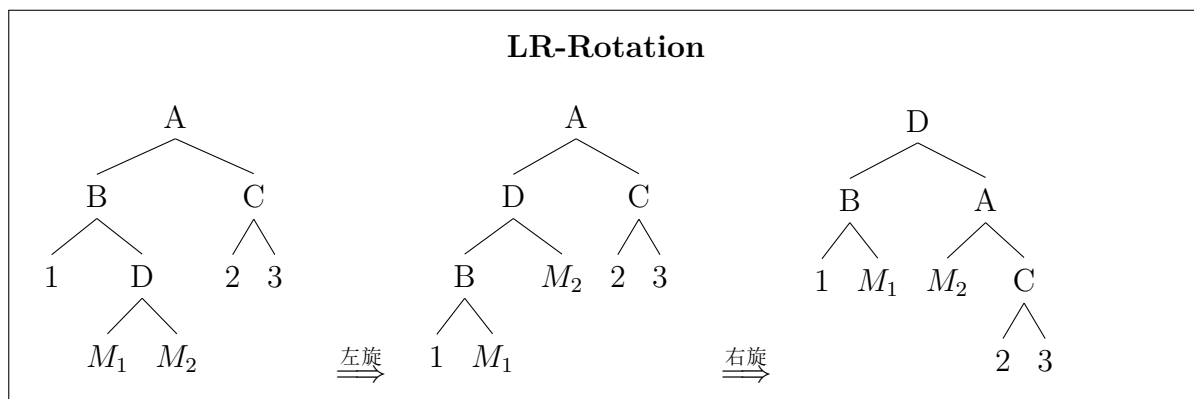
- 左子树和右子树的高度差不超过 1。
- 对于每个节点，其左子树中的所有键值都小于该节点的键值，右子树中的所有键值都大于该节点的键值。
- 每个子树都是 AVL 树。
- AVL 树的插入和删除操作都会涉及到旋转操作，旋转操作分为左旋和右旋。左旋和右旋都是以某个节点为支点，将该节点和其子树进行旋转，以达到平衡的目的。

AVL 树的自平衡基于两个基本操作，由于树中节点排列顺序一定，所以可以通过修改父级节点中的指针来完成节点高度的提升和降低，这两个操作称为左旋 (L-Rotation) 和右旋 (R-Rotation)。考虑以下插入操作，其中 M 表示高度大于 2 的子树，阿拉伯数字表示空指针：

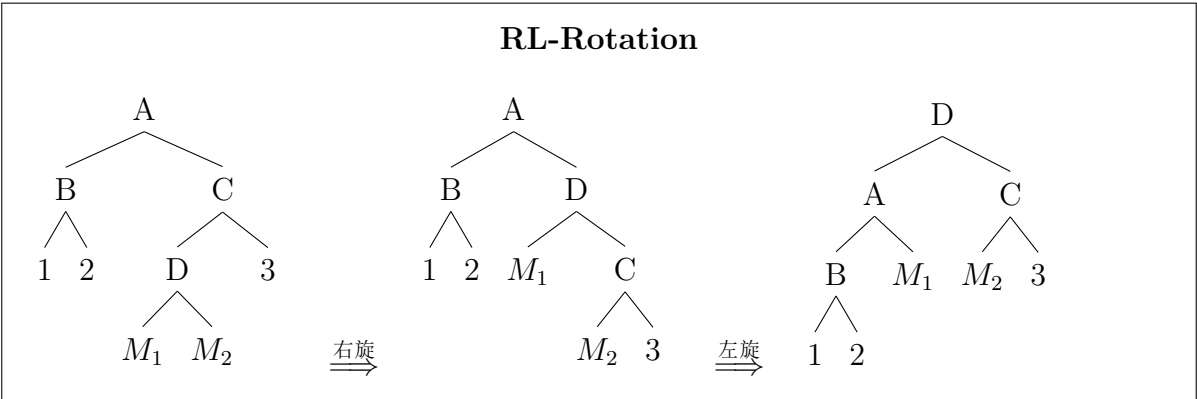


在这两个对称的操作中，以右旋为例，检测到 B 的高度差绝对值大于 1，于是将 B 节点提拔为根节点，其右节点指向它的父节点 A，A 的左指针则指向 B 原先的右指针指向的地址。这样一来就实现了树的平衡。

而对于 M 接在“中间”位置上的时候，情况则变得麻烦一点，但我们不难看出这时的平衡可以通过两次左右旋的交替来完成，这时子树在 M_1 或是在 M_2 上是等价的：



观察到先绕 B 进行左旋，再绕 A 进行了右旋，最终把 2 提拔为根节点，使 M_1 、 M_1 落在等价高度上，树恢复了平衡。注意在两个操作中，围绕的节点的子节点都是 2。同理：



AVL 树的查找操作和二叉搜索树一样，时间复杂度为 $O(\log n)$ 。由于 AVL 树的平衡性质，它的插入、删除、查找等操作都具有较稳定的时间复杂度。但是 AVL 树对于频繁的插入和删除操作，会频繁地进行旋转操作，导致效率下降，因此在这种情况下，可以使用其他自平衡树，如红黑树。

1.3.2 抽象数据类型

抽象数据类型（ADT）是一种数据类型的数学模型，其定义仅依赖于它的行为和特征，而不依赖于其具体的实现。它描述了一组数据和定义在这组数据上的一组操作，这些操作可以被认为是一个黑盒子，其内部的实现细节被隐藏。ADT 不考虑具体的实现方式，只关注数据的抽象行为和操作。

一个抽象数据类型通常包含三个方面：

- 1. 数据表示：定义 ADT 中的数据结构，通常用类或结构体来实现。
- 2. 数据操作：定义一组在数据结构上的操作，比如增加、删除、查找、修改等。
- 3. 约束条件：对数据的操作进行限制，以确保数据结构的正确性。

使用抽象数据类型可以帮助我们将程序设计的重点放在程序的功能实现上，而不用关注具体的实现方式。这也使得代码的维护和扩展变得更加容易，因为我们只需要关注 ADT 的定义和使用方法，而不用关心其具体实现细节。

比如，在本项目中，可以使用 AVL 树的抽象数据类型，在一个头文件 AVLTree.h 中定义 AVL 树的结构类型和操作函数原型，在 AVLTree.c 文件中编写树操作的函数，并通过 static 关键字将一些函数的作用域限定在一个文件内。还可以通过编写 Format.h 文件来定义通讯录数据类型，便于修改；编写 gui.c 文件提供给主文件调用的交互界面。

2 代码编写

2.1 定义类型

首先需要定义元素的类型：在本例中包括姓名、电话号码、住址、工作单位。然后是二叉节点的类型：每一个节点包含一个元素（Item 类型）和两个指向下级节点的指针，然后定义一个树结构（Tree）包含指向根节点的指针和树中元素的个数。还需要定义查找时使用的父子节点结构，定义为 Pair 结构，包含指向父节点和子节点的指针。

```

1  AVLTree.h -AVL树ADT原型
2
3  //二叉树节点结构体
4  typedef struct trnode
5  {
6      Item item;
7      int height;
8      struct trnode * left;
9      struct trnode * right;
10 } Trnode;
11
12 //二叉树访问结构体
13 typedef struct tree
14 {
15     Trnode * root;
16     int size;
17 } Tree;
18
19 //二叉查找结构体
20 typedef struct pair
21 {
22     Trnode * parent;
23     Trnode * child;
24 } Pair;

```

```

1  //Format.h -电话簿信息结构定义
2  typedef struct item
3  {
4      char name[MAXLINE];
5      char phone[PHONENUMBERLINE];
6      char workplace[MAXLINE];
7      char address[MAXLINE];
8  } Item;

```

2.2 建立接口

在编写程序之前, 先将 ADT 接口将要实现的功能抽象地写在一个头文件中, 并提供函数原型。

```

1  //AVLTree.h -AVL树ADT原型
2  //树中不允许重复数据
3  //左子树的所有项都在根节点的前面, 右子树的所有项都在根节点的后面
4
5  /*函数原型*/
6
7  //O: 将树初始化为空
8  //P: NULL
9  //E: 树被初始化为空, 并返回一个指向Tree的指针
10 Tree * InitializeTree(void);
11
12 //O: 确定树是否为空
13 //P: ptree指向一个Tree类型
14 //E: 若为空, 返回true, 否则返回false
15 bool IsTreeEmpty(const Tree * ptree);
16
17 //O: 确定树是否已满
18 //P: ptree指向一个Tree类型
19 //E: 若为空, 返回true, 否则返回false
20 bool IsTreeFull(const Tree * ptree);
21
22 //O: 确定树中的项数

```

```

23 //P:ptree指向一个Tree类型
24 //E:返回一个树的项数
25 int TreeItemCount(const Tree * ptree);
26
27 //O:在树中添加一个项
28 //P:pi指向待添加项的地址,ptree指向要添加到的树
29 //E:如果可以添加,则返回该节点及其父级
30 bool AddItem(const Item * pi, Tree * ptree);
31
32 //O:在树中删除一个项
33 //P:pi指向待删除项的地址,ptree指向要操作的树
34 //E:如果可以删除,则返回true,同时将该项从树中删除,否则返回false
35 bool DelItem(const Item * pi, Tree * ptree);
36
37 //O:在树中查找一个项
38 //P:pi指向待查找项的地址,ptree指向要操作的树
39 //E:如果查找到,则返回true,否则返回false
40 bool InTree(const Item * pi, const Tree * ptree);
41
42 //O:清空树
43 //P:ptree指向一个Tree类型
44 //E:ptree指向的Tree被清空
45 void DelAll(Tree * ptree);
46
47 //O:将通讯录写入指定文件流
48 //P:ptree指向一个Tree类型,File指向一个文件
49 //E:ptree指向的Tree中的每一个Item以指定顺序写入文件流
50 bool TreeWritef(const Tree * ptree, FILE * fp, const char * order);
51
52 //O:从指定文件流中读取一个树
53 //P:ptree指向一个Tree类型,File指向一个文件
54 //E:ptree指向的Tree中的每一个Item以指定顺序写入文件流并自平衡
55 bool TreeReadf(Tree * ptree, FILE * fp);
56
57 //O:在交互页面(标准输入流中)读取树的一个元素
58 //P:ptree指向一个Tree类型
59 //E:ptree指向的树中的Item写入文件流并自平衡
60 bool TreeReadOne(Tree * ptree);
61
62 //O:在输入流中显示二叉树结构
63 //P:ptree指向一个Tree类型,fp是一个文件指针
64 //E:ptree指向的树以图表方式显示在fp指向的文件流中,以name为标签
65 bool ShowTree(Tree *ptree, FILE * fp);
66
67 //O:在指定文件流中输出一个Item结构
68 //P:fp是一个文件指针
69 //E:在fp指向的文件中格式化打印了一个item中的数据
70 bool PrintItem(const Item item, FILE * fp);
71
72 //O:在树中搜索节点
73 //P:pi指向一个Item结构,ptree指向一个Tree结构
74 //O:值为pi指向的值的节点,返回以这个节点为子节点的Pair结构
75 Pair SeekItem(const Item * pi, const Tree * ptree);

```

2.3 实现接口

然后单独在 AVLTree.c 文件中编写树操作的函数。有一些函数是比较基本的二叉树函数，比如中序遍历、递归显示元素、判断树是否为空、删除树等，这里不全部详细展开，只来看几个比较重要的函数。

```

1 bool InTree(const Item * pi, const Tree * ptree)
2 {
3     return (SeekItem(pi, ptree).child == NULL) ? false : true;
4 }

```

```

5 //节点导航
6 static bool ToLeft(const Item * i1, const Item * i2)
7 {
8     int comp1;
9     if((comp1 = strcmp(i1->name, i2->name)) < 0)
10         return true;
11     else
12         return false;
13 }
14
15 static bool ToRight(const Item * i1, const Item * i2)
16 {
17     int comp1;
18     if((comp1 = strcmp(i1->name, i2->name)) > 0)
19         return true;
20     else
21         return false;
22 }
23
24 //搜索节点(递归方法)
25 Pair SeekItem(const Item * pi, const Tree * ptree)
26 {
27     Pair look;
28     look.parent = NULL;
29     look.child = ptree->root;
30     if(look.child == NULL)
31         return look;
32     else
33         return RecuSeek(pi, look);
34 }
35
36 static Pair RecuSeek(const Item * pi, Pair look)
37 {
38     if(look.child == NULL)
39         return look;
40     else if(ToLeft(pi, &(look.child->item)))
41     {
42         look.parent = look.child;
43         look.child = look.child->left;
44         return RecuSeek(pi, look);
45     }
46     else if(ToRight(pi, &(look.child->item)))
47     {
48         look.parent = look.child;
49         look.child = look.child->right;
50         return RecuSeek(pi, look);
51     }
52     else
53         return look;
54 }

```

这里主要的函数是 Intree(), 但它本身只有一行代码, 而把工作交给了调用的代码来完成操作。而调用的 SeekItem() 和 ToLeft() 等函数声明时均带上了 static 关键字, 这是为了将函数作用域限制在 AVLTree.c 内部, 只向外界展示 Intree() 函数, 避免误调用。

```

1 static Trnode * AddNode(Trnode * new_node, Trnode * root)
2 {
3     Pair current;
4     current.parent = root;
5     new_node->height = root->height + 1;
6     if(ToLeft(&new_node->item, &root->item))
7     {
8         if(root->left == NULL)
9         {
10             root->left = new_node;

```

```

11         current.child = root->left;
12     }
13     else
14     {
15         current.child = AddNode(new_node, root->left); //递归查找子树
16         current.parent->left = current.child;
17     }
18 }
19 else if(ToRight(&new_node->item, &root->item))
20 {
21     if(root->right == NULL)
22     {
23         root->right = new_node;
24         current.child = root->right;
25     }
26     else
27     {
28         current.child = AddNode(new_node, root->right); //递归查找子树
29         current.parent->right = current.child;
30     }
31 }
32 else
33 {
34     fprintf(stderr, "FAIL TO ADD A NODE");
35     exit(EXIT_FAILURE);
36 }
37 //添加完成后检测父节点高度因子,从下到上逐次旋转
38 //高度因子为左最长子树高度-右,若为正数则R或LR,若为负数则L或RL
39 return Rotation(current);
40 }
41
42 //执行旋转操作,返回新的根节点
43 static Trnode * Rotation(Pair current)
44 {
45     int Rotate = GetHeightFactor(current.parent);
46     if(Rotate > 1)
47     {
48         if(GetHeightFactor(current.child) < 0)
49             current.parent->left = LeftRotate(current.child);
50         return RightRotate(current.parent);
51     }
52     else if(Rotate < -1)
53     {
54         if(GetHeightFactor(current.child) > 0)
55             current.parent->right = RightRotate(current.child);
56         return LeftRotate(current.parent);
57     }
58     else
59         return current.parent;
60 }
61
62 static int GetHeight(const Trnode * root)
63 {
64     if(root == NULL)
65         return 0;
66     if(root->left == NULL && root->right == NULL)
67         return root->height;
68     else if(root->left == NULL && root->right != NULL)
69         return GetHeight(root->right);
70     else if(root->left != NULL && root->right == NULL)
71         return GetHeight(root->left);
72     else
73         return MAX(GetHeight(root->left), GetHeight(root->right));
74 }
75
76 static int GetHeightFactor(const Trnode * root)
77 {
78     if(root == NULL)
79         return 0;
80     else if(root->left == NULL && root->right != NULL)
81         return root->height - GetHeight(root->right);

```

```

82     else if(root->left != NULL && root->right == NULL)
83         return GetHeight(root->left) - root->height;
84     else
85         return GetHeight(root->left) - GetHeight(root->right);
86 }
87
88 static void ChangeHeight(Trnode * root, int val)
89 {
90     if(root != NULL)
91     {
92         ChangeHeight(root->left, val);
93         root->height += val;
94         ChangeHeight(root->right, val);
95     }
96 }
97
98 static Trnode * RightRotate(Trnode * root)
99 {
100     Trnode * new_root = NULL;
101     new_root = root->left;
102     //左子树高度全部减1,右子树和根节点高度全部加1
103     ChangeHeight(root->left, -1);
104     root->height += 1;
105     ChangeHeight(root->right, 1);
106     //移动指针
107     root->left = (new_root->right == NULL) ? NULL : new_root->right;
108     new_root->right = root;
109     return new_root;
110 }
111
112 static Trnode * LeftRotate(Trnode * root)
113 {
114     Trnode * new_root = NULL;
115     new_root = root->right;
116     //右子树高度全部减1,左子树和根节点高度全部加1
117     ChangeHeight(root->right, -1);
118     root->height += 1;
119     ChangeHeight(root->left, 1);
120     //移动指针
121     root->right = (new_root->left == NULL) ? NULL : new_root->left;
122     new_root->left = root;
123     return new_root;
124 }

```

以上是 AVL 树插入节点并通过旋转实现自平衡的代码。程序先将新节点插入到应有位置，然后逐层递归获得节点高度因子。若高度因子绝对值大于一，则调用局部函数 Rotation() 进行旋转，Rotation() 函数基于搜索到的 Pair 结构对节点、选择调用 LeftRotate() 和 RightRotate() 进行旋转。需要注意：如果要执行双旋，则需要对三层节点进行操作。所以 LeftRotate() 和 RightRotate() 应该是 Pair，用以在递归时传递节点信息。

```

1  bool TreeWritef(const Tree * ptree, FILE * fp, const char * order)
2  {
3      if(ptree == NULL)
4      {
5          return false;
6          fprintf(stderr, "TREE DOES NOT EXISTS");
7      }
8      if(strcmp(order, "DESC"))
9          return PrintDESC(ptree->root, fp);
10     if(strcmp(order, "ASC"))
11         return PrintASC(ptree->root, fp);
12     return false;
13 }
14

```



```

15 static bool PrintASC(const Trnode * root, FILE * fp)
16 {
17     if(root != NULL)
18     {
19         PrintASC(root->right, fp);
20         fprintf(fp, "Item:\n");
21         PrintItem(root->item, fp);
22         PrintASC(root->left, fp);
23     }
24 }
25
26 static bool PrintDESC(const Trnode * root, FILE * fp)
27 {
28     if(root != NULL)
29     {
30         PrintDESC(root->left, fp);
31         fprintf(fp, "Item:\n");
32         PrintItem(root->item, fp);
33         PrintDESC(root->right, fp);
34     }
35 }
36
37 bool TreeReadf(Tree * ptree, FILE * fp)
38 {
39     char line[MAXLINE + 50] = {'\0'}; //这里挺容易溢出的
40     Item item;
41     int count = 0;
42     while (fgets(line, sizeof(line), fp) != NULL)
43     { // 逐行读取文件内容
44         int i = 0;
45         while(line[i] == ' ' || line[i] == '\n' || line[i] == '\t')
46             ++i; //跳过空白符
47         if (line[i] == 'N' && line[i + 1] != 'o')
48             sscanf(line + i, "NAME: %s", item.name);
49         else if (line[i] == 'P')
50             sscanf(line + i, "PHONE: %s", item.phone);
51         else if (line[i] == 'W')
52             sscanf(line + i, "WORKPLACE: %s", item.workplace);
53         else if (line[i] == 'A')
54         {
55             sscanf(line + i, " ADDRESS: %s", item.address);
56             if(!AddItem(&item, ptree))
57                 return false;
58             ++count;
59         }
60         else
61             continue;
62     }
63     fprintf(stdout, "Success to read %d piece(s) from \"DATABASE\"\n", count);
64     return true;
65 }
66
67 bool TreeReadOne(Tree * ptree)
68 {
69     if(ptree == NULL)
70     {
71         return false;
72         fprintf(stderr, "TREE DOES NOT EXISTS");
73     }
74     Item * pi = (Item *) malloc(sizeof(Item));
75     fputs("Enter the item data.\"RE_\" to restart.\n", stdout);
76     const char *filed[] = {"EMPTY", "name", "phone", "workplace", "address"};
77     int i = 1;
78     for(i = 1; i <= 4; ++i)
79     {
80         fprintf(stdout, "%s:", *(filed+i));
81         char temp[MAXLINE] = {'\0'};
82         fscanf(stdin, "%s", temp);
83         if(strcmp(temp, "RE_") == 0)
84         {
85             i = 0;

```

```

86         continue;
87     }
88     if(i == 1)
89         strcpy(pi->name, temp);
90     if(i == 2)
91         strcpy(pi->phone, temp);
92     if(i == 3)
93         strcpy(pi->workplace, temp);
94     if(i == 4)
95         strcpy(pi->address, temp);
96 }
97 int wrong = Islegal(pi);
98 if(wrong > 0)
99 {
100     fprintf(stdout, "ILLEGAL INPUT OF _%s_ TRY AGAIN\n\n", filed[wrong]);
101     TreeReadOne(ptree);
102 }
103 else if(!AddItem(pi, ptree))
104 {
105     fprintf(stderr, "FAIL TO INSERT\n");
106     free(pi);
107     return false;
108 }
109 free(pi);
110 fprintf(stdout, "\nDone!\n");
111 return true;
112 }
113
114 //检测输入是否合法,若有错误则返回错误项标号,否则返回0
115 static int Islegal(Item * pi)
116 {
117     if(strlen(pi->phone) != 11)
118         return 2;
119     for(int i = 0; i < 11; ++i)
120         if((pi->phone)[i] < '0' || (pi->phone)[i] > '9')
121             return 2;
122     return 0;
123     //目前只要求检测电话号码的合法性
124 }

```

题目要求需要从文件中读取并将记录写入文件，所以编写了 bool TreeWritf() 函数向指定文件流中写入联系人数据，并添加了可递增和递减显示的选项。这个函数向 stdout 写入时即可在屏幕终端显示。TreeReadf() 则用于从文件流中批量读取联系人信息；TreeReadOne() 则是从标准输入流中一次读取一个联系人的信息。

2.4 交互界面

最后在另外一个文件中调用显示页面函数 ShowGUI(), 并在 main() 中建立一个循环，以多次操作。

```

1 //TelephoneBook.c -程序主函数&图形化界面
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <ctype.h>
5 #include <string.h>
6 #include <stdbool.h>
7 #include "AVLTree.h"
8 #include "Format.h"
9 //#define DATABASE diyname
10
11 int main(void)//日后可加启动参数
12 {

```

```

13     printf("Initializing GUI.....Done!\n");
14
15     FILE * fpr = fopen(DATABASE, "r");
16     FILE * fpw = NULL;
17     Tree * ptree = InitializeTree();
18     char line[MAXLINE] = {'\0'};
19     if(!TreeReadf(ptree, fpr))
20     {
21         fprintf(stderr, "FAIL TO READ FROM FILE\n");
22         exit(EXIT_FAILURE);
23     }
24     int choice = 7;
25     while(true)
26     {
27         printf("\n\nPress Enter....");
28         getchar();
29         choice = ShowGUI();
30         if(choice == 0)
31         {
32             fprintf(stderr, "FAIL TO SHOW GUI\n");
33             exit(EXIT_FAILURE);
34         }
35         switch (choice)
36         {
37             case 1:
38                 if(!ShowAllContacts(ptree))
39                     printf("FAIL TO SHOW ALL CONTACTS");
40                 break;
41             case 2:
42                 printf("\nEnter the contact's phone to search: ");
43                 scanf("%s", line);getchar();
44                 if(IsContact(ptree->root, line, stdout) == 0)
45                     printf("\nNo data found.\n");
46                 break;
47             case 3:
48                 printf("\nEnter the contact's name to search: ");
49                 scanf("%s", line);getchar();
50                 UpdateContact(ptree, line);
51                 break;
52             case 4:
53                 printf("\nStart adding :\n");
54                 char ch = 'Y';
55                 while(true)
56                 {
57                     TreeReadOne(ptree);
58                     printf("\nContinue to add? (Y/N) :");
59                     getchar();scanf("%c", &ch);getchar();
60                     if(ch == 'Y')
61                         continue;
62                     else
63                         break;
64                 }
65                 break;
66             case 5:
67                 printf("\nEnter the contact's name to delete: ");
68                 scanf("%s", line);getchar();
69                 Item item;
70                 strcpy(item.name, line);
71                 Tnode *node;
72                 node = SeekItem(&item, ptree).child;
73                 if(node == NULL)
74                 {
75                     printf("\nFind no data.\n");
76                     break;
77                 }
78                 else
79                     DelItem(&(node->item), ptree);
80                 printf("\nDeleted.\n");
81                 break;
82             case 6:
83                 printf("\nCurrent number in phone book : %d \n", TreeItemCount(ptree));

```

```

84         break;
85     case 7:
86         fpw = fopen(DATABASE, "w");//清空文件
87         DelAll(ptree);
88         break;
89     case 8:
90         fpw = fopen(DATABASE, "w");
91         TreeWritef(ptree, fpw, "ASC");//默认升序写入
92         DelAll(ptree);
93         return 0;
94         break;
95     default:
96         printf("Wrong choice!Try again.\n");
97         break;
98     }
99 }
100 return 0;
101 }
102
103 static int ShowGUI(void)
104 {
105     printf("\n*****\n");
106     printf("* 1. Show all contacts' info * \n");
107     printf("* 2. Inquire contacts' info * \n");
108     printf("* 3. Update contacts' info * \n");
109     printf("* 4. Add a contacts * \n");
110     printf("* 5. Delete a contact * \n");
111     printf("* 6. Counting number of contacts * \n");
112     printf("* 7. Delete the telephone book * \n");
113     printf("* 8. Exit * \n");
114     printf("*****\n");
115     printf("Enter a number to make your choice: ");
116
117     int n;
118     scanf("%d", &n);
119     return n;
120 }

```

这里的 DATABASE 已经在 AVLTree.h 中宏定义为“data.txt”，也就是默认将数据储存在一个名为“data.txt”的文件中。在主文件中可以重新定义并修改。

3 测试与运行

在命令提示符中输入

```
gcc -o AVLTelephoneBook AVLTree.c TelephoneBook.c
```

进行多文件编译，得到了可执行文件 AVLTelephoneBook.exe。
在此之前准备了数据文件“data.txt”，内容如下：

```

1  Item:
2  NAME:   pyl
3  PHONE:  18177965656
4  WORKPLACE: sb
5  ADDRESS: SB
6
7  Item:
8  NAME:   lwj
9  PHONE:  14191919198

```

```
10      WORKPLACE: 15
11      ADDRESS: 5
12
13      Item:
14      NAME: cxy
15      PHONE: 19177956739
16      WORKPLACE: 114
17      ADDRESS: 514
```

打开 AVLTelephoneBook.exe，可见

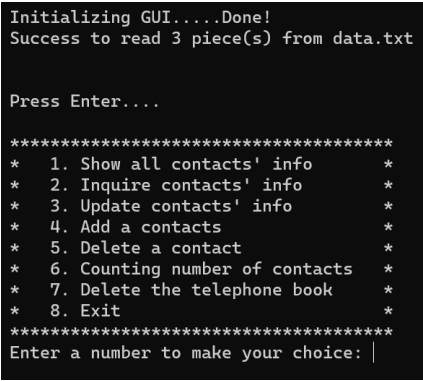


图 1: 初始化界面

初始化成功并显示成功读取数据。接下来测试读取数据是否正确。

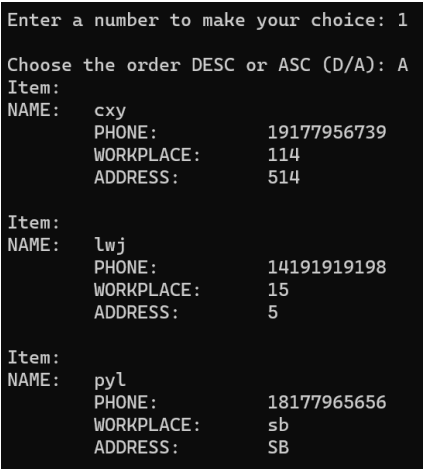


图 2: 采用字母升序（ASC）输出电话簿

本次操作结束，返回主界面测试查询功能。

```

Press Enter...

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
Enter a number to make your choice: 2

Enter the contact's phone to search: 19177956739
NAME: cxy
PHONE: 19177956739
WORKPLACE: 114
ADDRESS: 514

Press Enter...|

```

图 3: 成功根据电话查询联系人

测试修改联系人信息，按姓名检索：

```

Press Enter...

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
Enter a number to make your choice: 3

Enter the contact's name to search: pyl

new:
Enter the item data."RE_" to restart.
name:PYL
phone:11451419198
workplace:23
address:14

Done!

```

图 4: 执行修改联系人信息，提示成功

```

Item:
NAME:   PYL
PHONE:  11451419198
WORKPLACE: 23
ADDRESS: 14

```

图 5: 更新后的联系人信息

测试插入联系人信息：可随时在字段中输入“RE_”来重置添加；添加时会检查元素合法性，本例中只需要检查电话号码是否为 11 位纯数字。

```

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
Enter a number to make your choice: 4

Start adding :
Enter the item data."RE_" to restart.
name:curren-chan
phone:55555555555
workplace:toresen-gakuen
address:mihono

Done!

Continue to add? (Y/N) :Y
Enter the item data."RE_" to restart.
name:admire-vega
phone:19119119191
workplace:toresen-gakuen
address:ritto

Done!

Continue to add? (Y/N) :N

Press Enter...|

```

图 6: 成功连续添加

```

Start adding :
Enter the item data."RE_" to restart.
name:meisyo-dotou
phone:191
workplace:RE_
name:meisyo-dotou
phone:16161161161
workplace:toresen-gakuen
address:ritto

Done!

Continue to add? (Y/N) :Y
Enter the item data."RE_" to restart.
name:speciall-week
phone:6363636
workplace:15
address:14
ILLEGAL INPUT OF _phone_ TRY AGAIN

Enter the item data."RE_" to restart.
name:|

```

图 7: 两种错误情况

然后测试删除联系人操作：

```
Press Enter....

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
Enter a number to make your choice: 5

Enter the contact's name to delete: lwj

Deleted.

Press Enter....

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
```

图 8: 显示成功删除 “lwj”

```
Choose the order DESC or ASC (D/A): D
Item:
NAME:      meisyo-dotou
PHONE:      16161161161
WORKPLACE:  toresen-gakuen
ADDRESS:    ritto

Item:
NAME:      cxy
PHONE:      19177956739
WORKPLACE:  114
ADDRESS:    514

Item:
NAME:      curren-chan
PHONE:      55555555555
WORKPLACE:  toresen-gakuen
ADDRESS:    mihono

Item:
NAME:      admire-vega
PHONE:      19119119191
WORKPLACE:  toresen-gakuen
ADDRESS:    ritto

Item:
NAME:      PYL
PHONE:      11451419198
WORKPLACE:  23
ADDRESS:    14

Item:
NAME:      11
PHONE:      11111111111
WORKPLACE:  111
ADDRESS:    111
```

图 9: 删除后的元素列表

元素计数：

```
Press Enter....

*****
* 1. Show all contacts' info *
* 2. Inquire contacts' info *
* 3. Update contacts' info *
* 4. Add a contacts *
* 5. Delete a contact *
* 6. Counting number of contacts *
* 7. Delete the telephone book *
* 8. Exit *
*****
Enter a number to make your choice: 6

Current number in phone book : 6
```

图 10: 正确显示当前树中有 6 个元素

选项 7 和选项 8 的区别在于：选项 7 会直接清空程序缓存中的树中的所有数据，然后回到主界面；而选项 8 会现将现有数据写入 DATABASE 指向的文件后退出程序，代码是：

```
1  ...
2  case 7:
3      fpw = fopen(DATABASE, "w");//清空文件
4      DelAll(ptree);
5      break;
6  case 8:
7      fpw = fopen(DATABASE, "w");
8      TreeWritef(ptree, fpw, "ASC");//默认升序写入
```

```

9         DelAll(ptree);
10        return 0;
11        break;
12    ...

```

在这里只演示选项 8，程序立即退出。打开 data.txt 文件

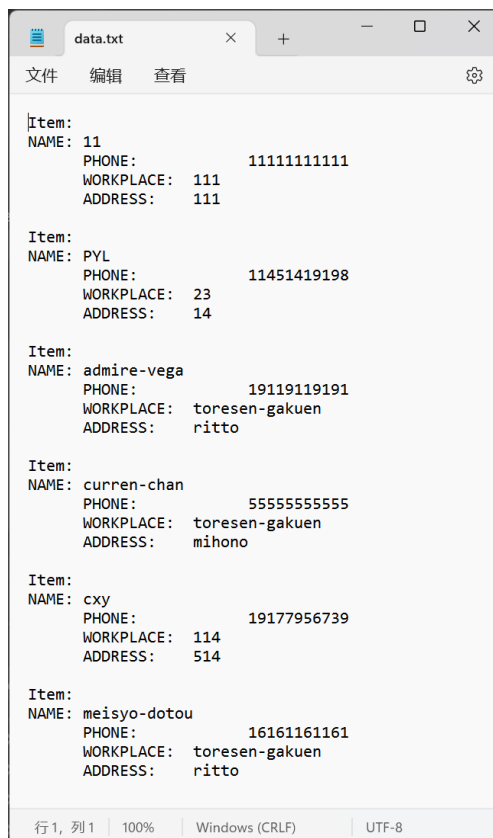


图 11: 数据被正确记录并写入文件

注意，必须执行选项 8 后才能将程序中操作的元素写入文件，否则文件的内容不会被修改。

4 完整代码实现

```

1  //AVLTree.h -AVL 树 ADT 原型
2  //树中不允许重复数据
3  //左子树的所有项都在根节点的前面,右子树的所有项都在根节点的后面
4  #pragma once
5  #include <stdbool.h>
6  #include <stdio.h>
7  #include "Format.h"
8
9  #ifndef MAXITEM
10 #define MAXITEM (50) //树中最多元素个数
11 #endif
12
13 #ifndef DATABASE
14 #define DATABASE "data.txt" //树中最多元素个数
15 #endif
16
17 /*定义结构*/
18

```



```

19 //二叉树节点结构体
20 typedef struct trnode
21 {
22     Item item;
23     int height;
24     struct trnode * left;
25     struct trnode * right;
26 } Trnode;
27
28 //二叉树访问结构体
29 typedef struct tree
30 {
31     Trnode * root;
32     int size;
33 } Tree;
34
35 //二叉查找结构体
36 typedef struct pair
37 {
38     Trnode * parent;
39     Trnode * child;
40 } Pair;
41
42 /*函数原型*/
43
44 //O:将树初始化为空
45 //P:NULL
46 //E:树被初始化为空,并返回一个指向Tree的指针
47 Tree * InitializeTree(void);
48
49 //O:确定树是否为空
50 //P:ptree指向一个Tree类型
51 //E:若为空,返回true,否则返回false
52 bool IsTreeEmpty(const Tree * ptree);
53
54 //O:确定树是否已满
55 //P:ptree指向一个Tree类型
56 //E:若为空,返回true,否则返回false
57 bool IsTreeFull(const Tree * ptree);
58
59 //O:确定树中的项数
60 //P:ptree指向一个Tree类型
61 //E:返回一个树的项数
62 int TreeItemCount(const Tree * ptree);
63
64 //O:在树中添加一个项
65 //P:pi指向待添加项的地址,ptree指向要添加到的树
66 //E:如果可以添加,则返回该节点及其父级
67 bool AddItem(const Item * pi, Tree * ptree);
68
69 //O:在树中删除一个项
70 //P:pi指向待删除项的地址,ptree指向要操作的树
71 //E:如果可以删除,则返回true,同时将该项从树中删除,否则返回false
72 bool DelItem(const Item * pi, Tree * ptree);
73
74 //O:在树中查找一个项
75 //P:pi指向待查找项的地址,ptree指向要操作的树
76 //E:如果查找到,则返回true,否则返回false
77 bool InTree(const Item * pi, const Tree * ptree);
78
79 //O:清空树
80 //P:ptree指向一个Tree类型
81 //E:ptree指向的Tree被清空
82 void DelAll(Tree * ptree);
83
84 //O:将通讯录写入指定文件流
85 //P:ptree指向一个Tree类型,File指向一个文件
86 //E:ptree指向的Tree中的每一个Item以指定顺序写入文件流
87 bool TreeWritef(const Tree * ptree, FILE * fp, const char * order);
88
89 //O:从指定文件流中读取一个树

```

```

90 //P:ptree指向一个Tree类型,File指向一个文件
91 //E:ptree指向的Tree中的每一个Item以指定顺序写入文件流并自平衡
92 bool TreeReadf(Tree * ptree, FILE * fp);
93
94 //O:在交互页面(标准输入流中)读取树的一个元素
95 //P:ptree指向一个Tree类型
96 //E:ptree指向的树中的Item写入文件流并自平衡
97 bool TreeReadOne(Tree * ptree);
98
99 //O:在输入流中显示二叉树结构
100 //P:ptree指向一个Tree类型,fp是一个文件指针
101 //E:ptree指向的树以图表方式显示在fp指向的文件流中,以name为标签
102 bool ShowTree(Tree *ptree, FILE * fp);
103
104 //O:在指定文件流中输出一个Item结构
105 //P:fp是一个文件指针
106 //E:在fp指向的文件中格式化打印了一个item中的数据
107 bool PrintItem(const Item item, FILE * fp);
108
109 //O:在树中搜索节点
110 //P:pi指向一个Item结构,ptree指向一个Tree结构
111 //O:值为pi指向的值的节点,返回以这个节点为子节点的Pair结构
112 Pair SeekItem(const Item * pi, const Tree * ptree);

```

```

1 //Format.h -电话簿信息结构定义
2 #pragma once
3 #define MAXLINE 20
4 #define PHONENUMBERLINE 11 + 2
5 //#define _DEBUG_MODE_ _DEBUG_MODE_ //debug模式开关-只使用name字段
6
7 #ifndef _DEBUG_MODE_
8 typedef struct item
9 {
10     char name[MAXLINE];
11     char phone[PHONENUMBERLINE];
12     char workplace[MAXLINE];
13     char address[MAXLINE];
14 } Item;
15
16 #else
17 typedef struct item
18 {
19     char name[MAXLINE];
20 } Item;
21
22 #endif

```

```

1 //AVLTree.c -AVL树ADT基本代码
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5 #include <ctype.h>
6 #include "AVLTree.h"
7 #include "Format.h"
8
9 ///////////////
10
11 /*局部函数声明*/
12 static bool ToLeft(const Item * i1, const Item * i2);
13 static bool ToRight(const Item * i1, const Item * i2);
14 static Pair RecuSeek(const Item * pi, Pair look);
15 static Trnode * MakeNode(const Item * pi);

```

```

16 static Trnode * AddNode(Trnode * new_node, Trnode * root);
17 static Trnode * Rotation(Pair current);
18 static int GetHeight(const Trnode * root);
19 static int GetHeightFactor(const Trnode * root);
20 static void ChangeHeight(Trnode * root, int val);
21 static Trnode * RightRotate(Trnode * root);
22 static Trnode * LeftRotate(Trnode * root);
23 static void InOrder(const Trnode * parent, void(*pfun)(Item item));
24 static void DelAllNodes(Trnode * root);
25 static void DeleteNode(Trnode **ptr);
26 static int Islegal(Item * pi);
27 static bool PrintDESC(const Trnode * root, FILE * fp);
28 static bool PrintASC(const Trnode * root, FILE * fp);
29 #define MAX(a, b) (a) > (b) ? (a) : (b)
30 /*函数代码*/
31
32 Tree * InitializeTree(void)
33 {
34     Tree *ptree;
35     ptree = (Tree *) malloc(1 * sizeof(Tree));
36     ptree->root = NULL;
37     ptree->size = 0;
38     return ptree;
39 }
40
41 bool IsTreeEmpty(const Tree * ptree)
42 {
43     if(ptree->root == NULL)
44         return true;
45     else
46         return false;
47 }
48
49 bool IsTreeFull(const Tree * ptree)
50 {
51     if(ptree->size >= MAXITEM)
52         return true;
53     else
54         return false;
55 }
56
57 int TreeItemCount(const Tree * ptree)
58 {
59     return ptree->size;
60 }
61
62 bool InTree(const Item * pi, const Tree * ptree)
63 {
64     return (SeekItem(pi, ptree).child == NULL) ? false : true;
65 }
66
67 //节点导航
68 static bool ToLeft(const Item * i1, const Item * i2)
69 {
70     int comp1;
71     if((comp1 = strcmp(i1->name, i2->name)) < 0)
72         return true;
73     else
74         return false;
75 }
76
77 static bool ToRight(const Item * i1, const Item * i2)
78 {
79     int comp1;
80     if((comp1 = strcmp(i1->name, i2->name)) > 0)
81         return true;
82     else
83         return false;
84 }
85
86 //搜索节点(递归方法)

```

```

87 Pair SeekItem(const Item * pi, const Tree * ptree)
88 {
89     Pair look;
90     look.parent = NULL;
91     look.child = ptree->root;
92     if(look.child == NULL)
93         return look;
94     else
95         return RecuSeek(pi, look);
96 }
97
98 static Pair RecuSeek(const Item * pi, Pair look)
99 {
100     if(look.child == NULL)
101         return look;
102     else if(ToLeft(pi, &(amp;look.child->item)))
103     {
104         look.parent = look.child;
105         look.child = look.child->left;
106         return RecuSeek(pi, look);
107     }
108     else if(ToRight(pi, &(amp;look.child->item)))
109     {
110         look.parent = look.child;
111         look.child = look.child->right;
112         return RecuSeek(pi, look);
113     }
114     else
115         return look;
116 }
117
118 //插入节点
119 bool AddItem(const Item * pi, Tree * ptree)
120 {
121     Trnode * new_node;
122     if(IsTreeFull(ptree))
123     {
124         fprintf(stderr, "MEMORY FULL\n");
125         return false;
126     }
127     if(SekItem(pi, ptree).child != NULL)
128     {
129         fprintf(stderr, "DUPLICATE DATE\n");
130         return false;
131     }
132     //用MakeNode函数创建节点并将new_node指向新节点
133     new_node = MakeNode(pi);
134     if(new_node == NULL)
135     {
136         fprintf(stderr, "CANNOT CRATE A NODE");
137         return false;
138     }
139     if(ptree->root == NULL)
140         ptree->root = new_node;
141     else
142         ptree->root = AddNode(new_node, ptree->root);
143     ptree->size++;
144     return true;
145 }
146
147 static Trnode * MakeNode(const Item * pi)
148 {
149     Trnode * new_node;
150     new_node = (Trnode *) malloc(sizeof(Trnode));
151     if(new_node != NULL)
152     {
153         new_node->item = *pi;
154         new_node->left = NULL;
155         new_node->right = NULL;
156         new_node->height = 0; //节点高度初始化为0
157     }

```

```

158     return new_node;
159 }
160
161 static Trnode * AddNode(Trnode * new_node, Trnode * root)
162 {
163     Pair current;
164     current.parent = root;
165     new_node->height = root->height + 1;
166     if(ToLeft(&new_node->item, &root->item))
167     {
168         if(root->left == NULL)
169         {
170             root->left = new_node;
171             current.child = root->left;
172         }
173         else
174         {
175             current.child = AddNode(new_node, root->left); //递归查找子树
176             current.parent->left = current.child;
177         }
178     }
179     else if(ToRight(&new_node->item, &root->item))
180     {
181         if(root->right == NULL)
182         {
183             root->right = new_node;
184             current.child = root->right;
185         }
186         else
187         {
188             current.child = AddNode(new_node, root->right); //递归查找子树
189             current.parent->right = current.child;
190         }
191     }
192     else
193     {
194         fprintf(stderr, "FAIL TO ADD A NODE");
195         exit(EXIT_FAILURE);
196     }
197     //添加完成后检测父节点高度因子,从下到上逐次旋转
198     //高度因子为左最长子树高度-右,若为正数则R或LR,若为负数则L或RL
199     return Rotation(current);
200 }
201
202 //执行旋转操作,返回新的根节点
203 static Trnode * Rotation(Pair current)
204 {
205     int Rotate = GetHeightFactor(current.parent);
206     if(Rotate > 1)
207     {
208         if(GetHeightFactor(current.child) < 0)
209             current.parent->left = LeftRotate(current.child);
210         return RightRotate(current.parent);
211     }
212     else if(Rotate < -1)
213     {
214         if(GetHeightFactor(current.child) > 0)
215             current.parent->right = RightRotate(current.child);
216         return LeftRotate(current.parent);
217     }
218     else
219         return current.parent;
220 }
221
222 static int GetHeight(const Trnode * root)
223 {
224     if(root == NULL)
225         return 0;
226     if(root->left == NULL && root->right == NULL)
227         return root->height;
228     else if(root->left == NULL && root->right != NULL)

```

```

229     return GetHeight(root->right);
230 else if(root->left != NULL && root->right == NULL)
231     return GetHeight(root->left);
232 else
233     return MAX(GetHeight(root->left), GetHeight(root->right));
234 }
235
236 static int GetHeightFactor(const Trnode * root)
237 {
238     if(root == NULL)
239         return 0;
240     else if(root->left == NULL && root->right != NULL)
241         return root->height - GetHeight(root->right);
242     else if(root->left != NULL && root->right == NULL)
243         return GetHeight(root->left) - root->height;
244     else
245         return GetHeight(root->left) - GetHeight(root->right);
246 }
247
248 static void ChangeHeight(Trnode * root, int val)
249 {
250     if(root != NULL)
251     {
252         ChangeHeight(root->left, val);
253         root->height += val;
254         ChangeHeight(root->right, val);
255     }
256 }
257
258 static Trnode * RightRotate(Trnode * root)
259 {
260     Trnode * new_root = NULL;
261     new_root = root->left;
262     //左子树高度全部减1,右子树和根节点高度全部加1
263     ChangeHeight(root->left, -1);
264     root->height += 1;
265     ChangeHeight(root->right, 1);
266     //移动指针
267     root->left = (new_root->right == NULL) ? NULL : new_root->right;
268     new_root->right = root;
269     return new_root;
270 }
271
272 static Trnode * LeftRotate(Trnode * root)
273 {
274     Trnode * new_root = NULL;
275     new_root = root->right;
276     //右子树高度全部减1,左子树和根节点高度全部加1
277     ChangeHeight(root->right, -1);
278     root->height += 1;
279     ChangeHeight(root->left, 1);
280     //移动指针
281     root->right = (new_root->left == NULL) ? NULL : new_root->left;
282     new_root->left = root;
283     return new_root;
284 }
285
286 //中序遍历-默认按首字母升序
287 static void InOrder(const Trnode * parent, void(*pfun)(Item item))
288 {
289     if(parent != NULL)
290     {
291         InOrder(parent->left, pfun);
292         (*pfun)(parent->item);
293         InOrder(parent->right, pfun);
294     }
295 }
296
297 //完全清空树
298 void DelAll(Tree * ptree)
299 {

```

```

300     if(ptree != NULL)
301         DelAllNodes(ptree->root);
302     else
303         return;
304     ptree->root = NULL;
305     ptree->size = 0;
306     free(ptree);
307 }
308
309 static void DelAllNodes(Tnode * root)
310 {
311     //中序遍历清空项
312     Tnode * pright;
313     if(root != NULL)
314     {
315         pright = root->right;
316         DelAllNodes(root->left);
317         free(root);
318         DelAllNodes(pright);
319     }
320 }
321
322 //删除某一元素
323 bool DelItem(const Item * pi, Tree * ptree)
324 {
325     Pair look;
326     look = SeekItem(pi, ptree);
327     if(look.child == NULL)
328         return false;
329     if(look.parent == NULL) //根节点情形
330         DeleteNode(&ptree->root);
331     else if(look.parent->left == look.child)
332         DeleteNode(&look.parent->left);
333     else
334         DeleteNode(&look.parent->right);
335     ptree->size--;
336     //检查是否平衡
337     if(Rotation(look) != NULL)
338         return true;
339     else
340         return false;
341 }
342
343 static void DeleteNode(Tnode **ptr)
344 {
345     Tnode * temp;
346     if((*ptr)->left == NULL)
347     {
348         temp = *ptr;
349         *ptr = (*ptr)->right;
350         free(temp);
351     }
352     else if((*ptr)->right == NULL)
353     {
354         temp = *ptr;
355         *ptr = (*ptr)->left;
356         free(temp);
357     }
358     else
359         //此时被删除的节点有两个子节点
360         //在被删除节点的右支树中找到最近的空位并连上去
361     {
362         for(temp = (*ptr)->left; temp->right != NULL; temp = temp->right)
363             continue;
364         temp->right = (*ptr)->right;
365         temp = *ptr;
366         *ptr = (*ptr)->left;
367         free(temp);
368     }
369 }
370

```

```

371 bool TreeWritef(const Tree * ptree, FILE * fp, const char * order)
372 {
373     if(ptree == NULL)
374     {
375         return false;
376         fprintf(stderr, "TREE DOES NOT EXISTS");
377     }
378     if(strcmp(order, "DESC"))
379         return PrintDESC(ptree->root, fp);
380     if(strcmp(order, "ASC"))
381         return PrintASC(ptree->root, fp);
382     return false;
383 }
384
385 static bool PrintASC(const Trnode * root, FILE * fp)
386 {
387     if(root != NULL)
388     {
389         PrintASC(root->right, fp);
390         fprintf(fp, "Item:\n");
391         PrintItem(root->item, fp);
392         PrintASC(root->left, fp);
393     }
394 }
395
396 static bool PrintDESC(const Trnode * root, FILE * fp)
397 {
398     if(root != NULL)
399     {
400         PrintDESC(root->left, fp);
401         fprintf(fp, "Item:\n");
402         PrintItem(root->item, fp);
403         PrintDESC(root->right, fp);
404     }
405 }
406
407 bool PrintItem(const Item item, FILE * fp)
408 {
409     fprintf(fp, "NAME:\t%s\n", item.name);
410     fprintf(fp, "\tPHONE:\t\t%s\n", item.phone);
411     fprintf(fp, "\tWORKPLACE:\t%s\n", item.workplace);
412     fprintf(fp, "\tADDRESS:\t%s\n\n", item.address);
413
414     return true;
415 }
416
417 bool TreeReadf(Tree * ptree, FILE * fp)
418 {
419     char line[MAXLINE + 50] = {'\0'}; //这里挺容易溢出的
420     Item item;
421     int count = 0;
422     while (fgets(line, sizeof(line), fp) != NULL)
423     {
424         // 逐行读取文件内容
425         int i = 0;
426         while(line[i] == ' ' || line[i] == '\n' || line[i] == '\t')
427             ++i; //跳过空白符
428         if (line[i] == 'N' && line[i + 1] != 'o')
429             sscanf(line + i, "NAME: %s", item.name);
430         else if (line[i] == 'P')
431             sscanf(line + i, "PHONE: %s", item.phone);
432         else if (line[i] == 'W')
433             sscanf(line + i, "WORKPLACE: %s", item.workplace);
434         else if (line[i] == 'A')
435         {
436             sscanf(line + i, " ADDRESS: %s", item.address);
437             if(!AddItem(&item, ptree))
438                 return false;
439             ++count;
440         }
441         else
442             continue;

```



```

442     }
443     fprintf(stdout, "Success to read %d piece(s) from \"DATABASE\\n\", count);
444     return true;
445 }
446
447 bool TreeReadOne(Tree * ptree)
448 {
449     if(ptree == NULL)
450     {
451         return false;
452         fprintf(stderr, "TREE DOES NOT EXISTS");
453     }
454     Item * pi = (Item *) malloc(sizeof(Item));
455     fputs("Enter the item data.\\\"RE_\\\" to restart.\\n\", stdout);
456     const char *filed[] = {"EMPTY", "name", "phone", "workplace", "address"};
457     int i = 1;
458     for(i = 1; i <= 4; ++i)
459     {
460         fprintf(stdout, \"%s:\", *(filed+i));
461         char temp[MAXLINE] = {'\\0'};
462         fscanf(stdin, \"%s\", temp);
463         if(strcmp(temp, "RE_") == 0)
464         {
465             i = 0;
466             continue;
467         }
468         if(i == 1)
469             strcpy(pi->name, temp);
470         if(i == 2)
471             strcpy(pi->phone, temp);
472         if(i == 3)
473             strcpy(pi->workplace, temp);
474         if(i == 4)
475             strcpy(pi->address, temp);
476     }
477     int wrong = Islegal(pi);
478     if(wrong > 0)
479     {
480         fprintf(stdout, "ILLEGAL INPUT OF _%s_ TRY AGAIN\\n\\n\", filed[wrong]);
481         TreeReadOne(ptree);
482     }
483     else if(!AddItem(pi, ptree))
484     {
485         fprintf(stderr, "FAIL TO INSERT\\n");
486         free(pi);
487         return false;
488     }
489     free(pi);
490     fprintf(stdout, "\\nDone!\\n");
491     return true;
492 }
493
494 //检测输入是否合法,若有错误则返回错误项标号,否则返回0
495 static int Islegal(Item * pi)
496 {
497     if(strlen(pi->phone) != 11)
498         return 2;
499     for(int i = 0; i < 11; ++i)
500         if((pi->phone)[i] < '0' || (pi->phone)[i] > '9')
501             return 2;
502     return 0;
503     //目前只要求检测电话号码的合法性
504 }

```

```

1 //TelephoneBook.c -程序主函数&图形化界面
2 #include <stdio.h>
3 #include <stdlib.h>

```

```

4  #include <ctype.h>
5  #include <string.h>
6  #include <stdbool.h>
7  #include "AVLTree.h"
8  #include "Format.h"
9  // #define DATABASE diyname
10 // 局部函数声明
11 static int ShowGUI(void);
12 static bool ShowAllContacts(Tree * ptree);
13 static bool UpdateContact(Tree * ptree, char *phone);
14 static int IsContact(const Trnode * root, char *name, FILE * fp);
15
16 int main(void) // 日后可加启动参数
17 {
18     printf("Initializing GUI.....Done!\n");
19
20     FILE * fpr = fopen(DATABASE, "r");
21     FILE * fpw = NULL;
22     Tree * ptree = InitializeTree();
23     char line[MAXLINE] = {'\0'};
24     if(!TreeReadf(ptree, fpr))
25     {
26         fprintf(stderr, "FAIL TO READ FROM FILE\n");
27         exit(EXIT_FAILURE);
28     }
29     int choice = 7;
30     while(true)
31     {
32         printf("\n\nPress Enter....");
33         getchar();
34         choice = ShowGUI();
35         if(choice == 0)
36         {
37             fprintf(stderr, "FAIL TO SHOW GUI\n");
38             exit(EXIT_FAILURE);
39         }
40         switch (choice)
41         {
42             case 1:
43                 if(!ShowAllContacts(ptree))
44                     printf("FAIL TO SHOW ALL CONTACTS");
45                 break;
46             case 2:
47                 printf("\nEnter the contact's phone to search: ");
48                 scanf("%s", line); getchar();
49                 if(IsContact(ptree->root, line, stdout) == 0)
50                     printf("\nNo data found.\n");
51                 break;
52             case 3:
53                 printf("\nEnter the contact's name to search: ");
54                 scanf("%s", line); getchar();
55                 UpdateContact(ptree, line);
56                 break;
57             case 4:
58                 printf("\nStart adding :\n");
59                 char ch = 'Y';
60                 while(true)
61                 {
62                     TreeReadOne(ptree);
63                     printf("\nContinue to add? (Y/N) :");
64                     getchar(); scanf("%c", &ch); getchar();
65                     if(ch == 'Y')
66                         continue;
67                     else
68                         break;
69                 }
70                 break;
71             case 5:
72                 printf("\nEnter the contact's name to delete: ");
73                 scanf("%s", line); getchar();
74                 Item item;

```

```

75         strcpy(item.name, line);
76         Tnode *node;
77         node = SeekItem(&item, ptree).child;
78         if(node == NULL)
79         {
80             printf("\nFind no data.\n");
81             break;
82         }
83         else
84             DelItem(&(amp;node->item), ptree);
85         printf("\nDeleted.\n");
86         break;
87     case 6:
88         printf("\nCurrent number in phone book : %d \n", TreeItemCount(ptree));
89         break;
90     case 7:
91         fpw = fopen(DATABASE, "w");//清空文件
92         DelAll(ptree);
93         break;
94     case 8:
95         fpw = fopen(DATABASE, "w");
96         TreeWritef(ptree, fpw, "ASC");//默认升序写入
97         DelAll(ptree);
98         return 0;
99         break;
100    case 114514:
101        ShowTree(ptree, stdout);
102        getchar();
103        break;
104    default:
105        printf("Wrong choice!Try again.\n");
106        break;
107    }
108    }
109    return 0;
110    }
111    static int ShowGUI(void)
112    {
113        printf("\n*****\n");
114        printf("*    1. Show all contacts' info      *\n");
115        printf("*    2. Inquire contacts' info      *\n");
116        printf("*    3. Update contacts' info      *\n");
117        printf("*    4. Add a contacts              *\n");
118        printf("*    5. Delete a contact            *\n");
119        printf("*    6. Counting number of contacts *\n");
120        printf("*    7. Delete the telephone book  *\n");
121        printf("*    8. Exit                       *\n");
122        printf("*****\n");
123        printf("Enter a number to make your choice: ");
124
125        int n;
126        scanf("%d", &n);
127        return n;
128    }
129    static bool ShowAllContacts(Tree * ptree)
130    {
131        char order = 0;
132        while(true)
133        {
134            printf("\nChoose the order DESC or ASC (D/A): ");
135            getchar();scanf("%c", &order);getchar();
136            if(order == 'D')
137            {
138                TreeWritef(ptree, stdout, "DESC");
139                return true;
140            }
141            else if(order == 'A')
142            {
143                TreeWritef(ptree, stdout, "ASC");
144                return true;
145            }

```

```

146         printf("WRONG ORDER (A/D)\n");
147     }
148     return false;
149 }
150
151 static int IsContact(const Trnode * root, char *phone, FILE * fp)
152 {
153     static int count = -1;
154     if(root != NULL)
155     {
156         IsContact(root->left, phone, fp);
157         if(!strcmp((root->item.phone), phone))
158             PrintItem(root->item, fp);
159         IsContact(root->right, phone, fp);
160     }
161     ++count;
162     return count;
163 }
164
165 static bool UpdateContact(Tree * ptree, char *name)
166 {
167     Item item;
168     strcpy(item.name, name);
169     Trnode *node = SeekItem(&item, ptree).child;
170     if(node == NULL)
171     {
172         printf("\nFind no data. Add a contact?");
173         return false;
174     }
175     else
176     {
177         DelItem(&(node->item), ptree);
178         printf("\nnew: \n");
179         TreeReadOne(ptree);
180     }
181     return true;
182 }

```

5 总结

通过这次大作业，我进一步掌握了 C 语言程序设计的基本语法、方法，能够做到熟练运用指针进行面向过程的编程，掌握了二叉树数据结构的处理和实现，并尝试利用抽象数据类型（ADT）来实现接口和封装，从而编写一个较为复杂的程序。

但在过程当中，也存在一些问题，例如接口中各函数通信效率低，有时不得不复用一大段代码，造成结构不够明确、语法不够简洁、代码可读性差。这或许也是 C 语言的一个弱点，就是难以很好地进行模块化设计——或者说“面向对象编程”。以后我会尝试用 C++ 将这些代码更细致、更有条理地封装在“类”之中。

同时我也发现对于一些比较复杂的数据结构，自己编写接口实现是比较困难的。这时就可以利用他人已经写好的代码，也就是“库”来帮助编程。这时库中的函数和方法对我们来说就是完全黑箱的“抽象数据类型”，我们只需调用即可。

编辑环境

代码编辑：Microsoft Visual Studio Code

编译器：gcc 6.4.0

文档编辑：L^AT_EX2