**北京科技大学实验报告**

学院：计通学院 专业：物联网工程 班级：物联1602

姓名：周康能 学号：41624657 实验日期： 2018年 6 月 11 日

**实验名称：《数据结构》实验4二叉树的应用**

**实验目的：掌握二叉树的链式存储结构，理解二叉排序树，熟悉树的基本操作，尤其是非递归遍历算法的应用。**

**实验内容：**

树表的查找：输入一个英文句子，按照字典顺序构造一棵二叉排序树；对此二叉排序树进行中序遍历，并将遍历序列输出到屏幕上。

要求：

（1）英文句子可从键盘输入，也可从txt文件输入；

（2）遍历算法采用非递归遍历算法；

（3）程序结束时需释放树空间。

**问题分析与算法思路：**

数据结构用到了二叉树和链式的堆栈。

输入可以选键盘或者txt，每一个句子都存在一个字符串数组里。

对于中序遍历，对于任意一个结点，优先访问左孩子，而左孩子结点又可以看成一个结点，继续访问其左孩子结点，直到空。空了才访问。

（1）若其左孩子不为空，P入栈，将其左孩子当成P，然后对P进行相同的处理。

（2）若其左孩子为空，则取栈顶元素并出栈，访问该栈顶元素，然后将当前P置为栈顶节点的右孩子

（3）直到P为NULL，栈为空，退出

用伪代码给出算法描述。

**最终报告中请删除此框！**

本程序时间复杂度为O(n)，空间复杂度合适。

**算法描述：**

typedef struct BinaryTreeNode

{

char m\_nString[10];

BinaryTreeNode\* m\_pLeft;

BinaryTreeNode\* m\_pRight;

}BinaryTreeNode,\*BinaryTreeList;

typedef struct stack

{

BinaryTreeNode \*pNode;

struct stack \*next;

}NodeStack,\*LinkStack;

此为二叉树和链栈的结构体定义

BinaryTreeNode\* DoneBinaryTree(char \*sentence)

{

for (int i = 0; i < 50; i++)

{

if (sentence[i] == ' ')

{

continue;

}

else if (

(sentence[i] >= 'a' && sentence[i] <= 'z')

||

(sentence[i] >= 'A' && sentence[i] <= 'Z')

)

{

//-----------------------------------------------------------------------------------------

//去一个单词

//-----------------------------------------------------------------------------------------

while (

//sentence[i] != ' ' || sentence[i] != '\0'

(sentence[i] >= 'a' && sentence[i] <= 'z')

||

(sentence[i] >= 'A' && sentence[i] <= 'Z')

)

{

mid[j] = sentence[i];

j++;

i++;

}

mid[j] = '\0'; //非常重要，字符串结尾

//-----------------------------------------------------------------------------------------

j = 0;

BinaryTreeNode \*pNode = CreateBinary(mid);

if (!pRoot) //无root，Node补上去

{

pRoot = pNode;

}

BinaryTreeNode \*midTree = pRoot;

while (midTree)

{

if (strcmp(midTree->m\_nString, pNode->m\_nString) == 1) //midTree大

{

if (!midTree->m\_pLeft) //无左子树，插入

{

midTree->m\_pLeft = pNode;

break;

}

else //否则左移到左子树

{

midTree = midTree->m\_pLeft;

}

}

else if (strcmp(midTree->m\_nString, pNode->m\_nString) == -1) //midTree小

{

if (!midTree->m\_pRight) //无右子树，插入

{

midTree->m\_pRight = pNode;

break;

}

else //否则右移到右子树

{

midTree = midTree->m\_pRight;

}

}

else

{

break;

}

}

}

else if (sentence[i] == '.') //取到.退出

{

break;

}

}

return pRoot;

}

此为插入数的函数

void main()

{

while (1)

{

printf("Enter 1 to use keyborad,2 to use txt.\n");

if (输入1 )

{

键盘读入

}

else if (输入2 )

{

文件读入

}

else

{

printf("Error! Please try again!\n");

continue;

}

}

BinaryTreeNode \*DTree;

DTree = DoneBinaryTree(sentence);

PrintTree(DTree);

infix\_Output(DTree); //输出出来

DestroyTree(DTree);

}

void DestroyTree(BinaryTreeNode\* pRoot)

{

if (pRoot != NULL)

{

BinaryTreeNode\* pleft = pRoot->m\_pLeft;

BinaryTreeNode\* pright = pRoot->m\_pRight;

delete pRoot; //C++Cdelete！

pRoot = NULL;

//DestroyTree(pRoot->m\_pLeft);

//DestroyTree(pRoot->m\_pRight);

DestroyTree(pleft);

DestroyTree(pright);

}

}

此为销毁树的函数

**程序实现：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright(c) :Ellis

\* All rights reserved.

\*

\* 文件名称:USTB2018\_data\_structure\_test4

\* 简要描述:二叉排序树的构建与遍历

\* 要求：输入一个英文句子，按照字典顺序构造一个二叉排序树

\* 采用非递归算法，对此二叉树进行中序遍历，打印遍历结果

\*

\*

\* 对于中序遍历，对于任意一个结点，优先访问左孩子，而左孩子结点又可以看成一个结点，继续访问其左孩子结点，直到空。

\* 空了才访问。

\*

\* （1）若其左孩子不为空，P入栈，将其左孩子当成P，然后对P进行相同的处理。

\* （2）若其左孩子为空，则取栈顶元素并出栈，访问该栈顶元素，然后将当前P置为栈顶节点的右孩子

\* （3）直到P为NULL，栈为空，退出

\*

\*

\* 创建日期:20180421\_20:55

\* 作者:Ellis

\* 说明:完成所有任务

\*

\* 修改日期:20180521\_00:21

\* 作者:Ellis

\* 说明:txt输入

\* 注意：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 宏定义、头文件、命名空间区

\* 创建日期: 20180421

\* 作者 : Ellis

\* 说明 : 无

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_CRT\_SECURE\_NO\_DEPRECATE //规避安全检查

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 结构体定义区

\* 创建日期: 20180421

\* 作者 : Ellis

\* 说明 : 二叉树+链栈

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct BinaryTreeNode

{

char m\_nString[10];

BinaryTreeNode\* m\_pLeft;

BinaryTreeNode\* m\_pRight;

}BinaryTreeNode,\*BinaryTreeList;

typedef struct stack

{

BinaryTreeNode \*pNode;

struct stack \*next;

}NodeStack,\*LinkStack;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 函数定义区

\* 创建日期: 20180421

\* 作者 : Ellis

\* 说明 : 定义各种函数,集成化，容易看

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BinaryTreeNode \*CreateBinary(char \*string); //创建二叉树

BinaryTreeNode\* DoneBinaryTree(char \*sentence); //构造搜索树

LinkStack CreatStack(); //定义栈

void Push(LinkStack top, BinaryTreeNode \*mid); //出栈

void Pop(LinkStack top); //入栈

BinaryTreeNode\* getTop(LinkStack top); //取栈顶

void infix\_Output(BinaryTreeNode \*pRoot); //中序遍历

void PrintTreeNode(BinaryTreeNode\* pNode); //打印结点

void PrintTree(BinaryTreeNode\* pRoot); //打印树

void DestroyTree(BinaryTreeNode\* pRoot); //销毁树

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void main()

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

char sentence[50];

while (1)

{

printf("Enter 1 to use keyborad,2 to use txt.\n");

char d;

d = getchar();

getchar();

if (d == '1' )

{

gets(sentence);

break;

}

else if (d =='2' )

{

FILE \*file;

char name\_of\_txt[20];

printf("Please enter the name of the txt...\n");

scanf("%s", name\_of\_txt);

file = fopen(name\_of\_txt, "r");

if (!file)

{

printf("Could not open this txt!\n");

system("pause");

}

else

{

printf("Open the txt successfully!\n");

}

char ch;

int k = 0;

while (

(ch = fgetc(file)) != EOF //逐个字符去读取

)

{

sentence[k] = ch;

k++;

ch = NULL;

}

sentence[k] = '\0';

break;

}

else

{

printf("Error! Please try again!\n");

continue;

}

}

BinaryTreeNode \*DTree;

DTree = DoneBinaryTree(sentence);

PrintTree(DTree);

infix\_Output(DTree);

DestroyTree(DTree);

printf("\n");

system("pause");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:BinaryTreeNode \*CreateBinary(char \*string)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BinaryTreeNode \*CreateBinary(char \*string)

{

BinaryTreeNode \*pNode = (BinaryTreeList)malloc(sizeof(BinaryTreeNode));

pNode->m\_pLeft = NULL;

pNode->m\_pRight = NULL;

strcpy(pNode->m\_nString, string);

return pNode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:BinaryTreeNode\* DoneBinaryTree(char \*sentence)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BinaryTreeNode\* DoneBinaryTree(char \*sentence)

{

char mid[10];

//memset(mid, '0', sizeof(mid));

int j = 0;

BinaryTreeNode \*pRoot = NULL;

for (int i = 0; i < 50; i++)

{

if (sentence[i] == ' ')

{

continue;

}

else if (

(sentence[i] >= 'a' && sentence[i] <= 'z')

||

(sentence[i] >= 'A' && sentence[i] <= 'Z')

)

{

//-----------------------------------------------------------------------------------------

//去一个单词

//-----------------------------------------------------------------------------------------

while (

//sentence[i] != ' ' || sentence[i] != '\0'

(sentence[i] >= 'a' && sentence[i] <= 'z')

||

(sentence[i] >= 'A' && sentence[i] <= 'Z')

)

{

mid[j] = sentence[i];

j++;

i++;

}

mid[j] = '\0'; //非常重要，字符串结尾

//-----------------------------------------------------------------------------------------

j = 0;

BinaryTreeNode \*pNode = CreateBinary(mid);

if (!pRoot) //无root，Node补上去

{

pRoot = pNode;

}

BinaryTreeNode \*midTree = pRoot;

while (midTree)

{

if (strcmp(midTree->m\_nString, pNode->m\_nString) == 1) //midTree大

{

if (!midTree->m\_pLeft) //无左子树，插入

{

midTree->m\_pLeft = pNode;

break;

}

else //否则左移到左子树

{

midTree = midTree->m\_pLeft;

}

}

else if (strcmp(midTree->m\_nString, pNode->m\_nString) == -1) //midTree小

{

if (!midTree->m\_pRight) //无右子树，插入

{

midTree->m\_pRight = pNode;

break;

}

else //否则右移到右子树

{

midTree = midTree->m\_pRight;

}

}

else

{

break;

}

}

}

else if (sentence[i] == '.') //取到.退出

{

break;

}

}

return pRoot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:LinkStack CreatStack()

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LinkStack CreatStack()

{

LinkStack top = (LinkStack)malloc(sizeof(NodeStack));

top->next = NULL;

top->pNode = NULL;

return top;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void Push(LinkStack top, BinaryTreeNode \*mid)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Push(LinkStack top, BinaryTreeNode \*mid)

{

LinkStack node = (LinkStack)malloc(sizeof(NodeStack));

node->pNode = mid;

node->next = top->next;

top->next = node;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void Pop(LinkStack top)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Pop(LinkStack top)

{

if (!top->next)

{

return;

}

else

{

LinkStack temp = top->next;

top->next = temp->next;

free(temp);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:BinaryTreeNode\* getTop(LinkStack top)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BinaryTreeNode\* getTop(LinkStack top)

{

if (!top->next)

{

return 0;

}

else

{

return top->next->pNode;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void infix\_Output(BinaryTreeNode \*pRoot)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void infix\_Output(BinaryTreeNode \*pRoot)

{

//LinkStack S = NULL;

LinkStack top = CreatStack();

BinaryTreeNode\* p = pRoot;

while (1)

{

while (p)

{

Push(top, p);

p = p->m\_pLeft;

}

if (!top->next)

{

break;

}

p = getTop(top);

Pop(top);

printf("%s\t", p->m\_nString);

p = p->m\_pRight;

}

//（1）若其左孩子不为空，P入栈，将其左孩子当成P

//（2）若其左孩子为空，则取栈顶元素并出栈，访问该栈顶元素，然后将当前P置为栈顶节点的右孩子

//（3）直到P为NULL，栈为空，退出

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void PrintTreeNode(BinaryTreeNode\* pNode)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PrintTreeNode(BinaryTreeNode\* pNode)

{

if (pNode != NULL)

{

printf("The value of this node is %s\n", pNode->m\_nString);

if (pNode->m\_pLeft != NULL)

{

printf("The value of its left child is %s\n", pNode->m\_pLeft->m\_nString);

}

else

{

printf("The value doesn't exist.Its left child is Null.\n");

}

if (pNode->m\_pRight != NULL)

{

printf("The value of its right child is %s\n", pNode->m\_pRight->m\_nString);

}

else

{

printf("The value doesn't exist.Its right child is Null.\n");

}

}

else

{

printf("The node is Null!\n");

}

printf("\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void PrintTree(BinaryTreeNode\* pRoot)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PrintTree(BinaryTreeNode\* pRoot)

{

PrintTreeNode(pRoot);

if (pRoot != NULL)

{

if (pRoot->m\_pLeft != NULL)

{

PrintTree(pRoot->m\_pLeft);

}

if (pRoot->m\_pRight != NULL)

{

PrintTree(pRoot->m\_pRight);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void PrintTree(BinaryTreeNode\* pRoot)

Author :Ellis

Date :20180421

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DestroyTree(BinaryTreeNode\* pRoot)

{

if (pRoot != NULL)

{

BinaryTreeNode\* pleft = pRoot->m\_pLeft;

BinaryTreeNode\* pright = pRoot->m\_pRight;

delete pRoot; //C++Cdelete！

pRoot = NULL;

//DestroyTree(pRoot->m\_pLeft);

//DestroyTree(pRoot->m\_pRight);

DestroyTree(pleft);

DestroyTree(pright);

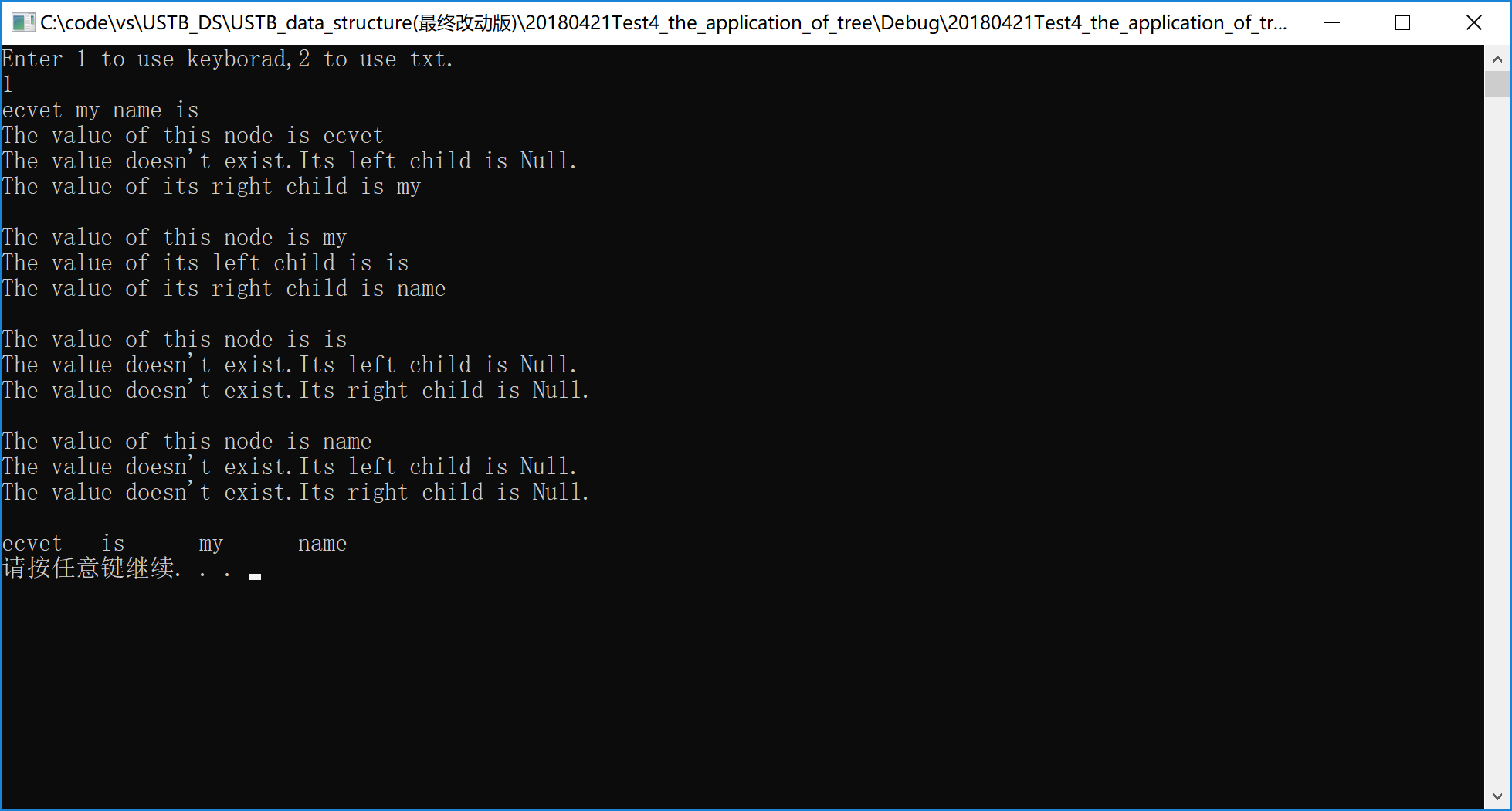
}

}

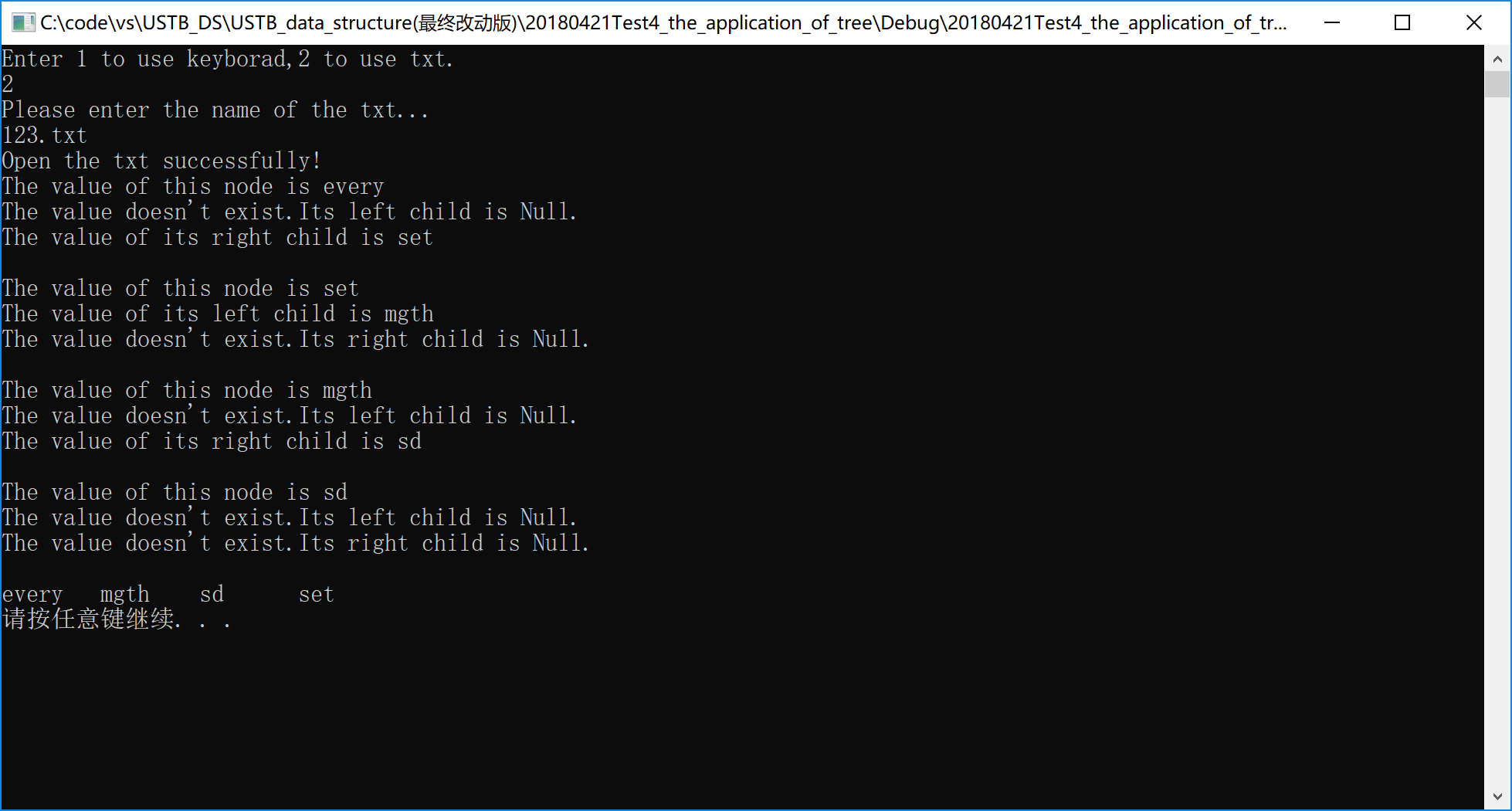
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

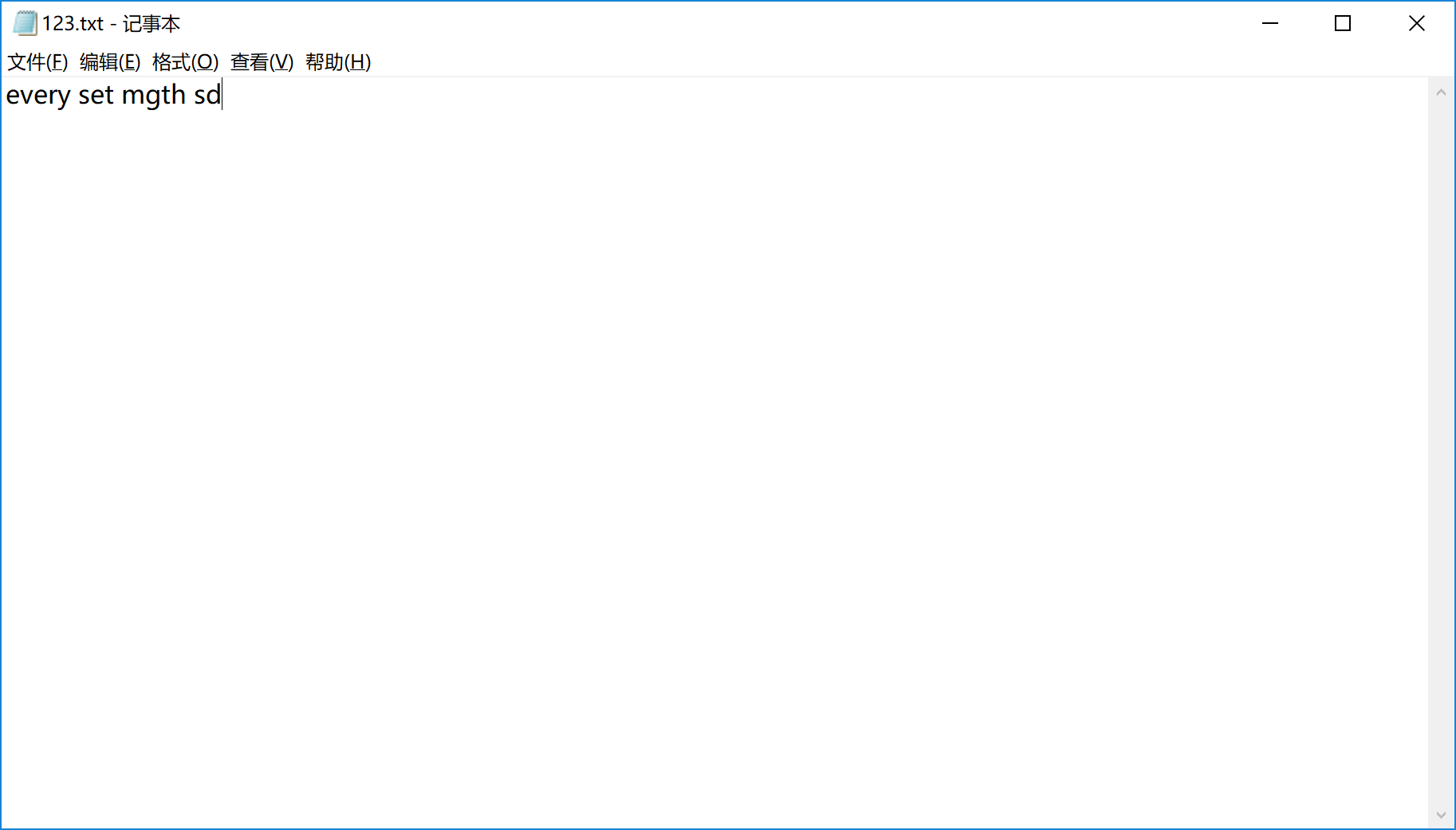
**测试：**

测试键盘输入：

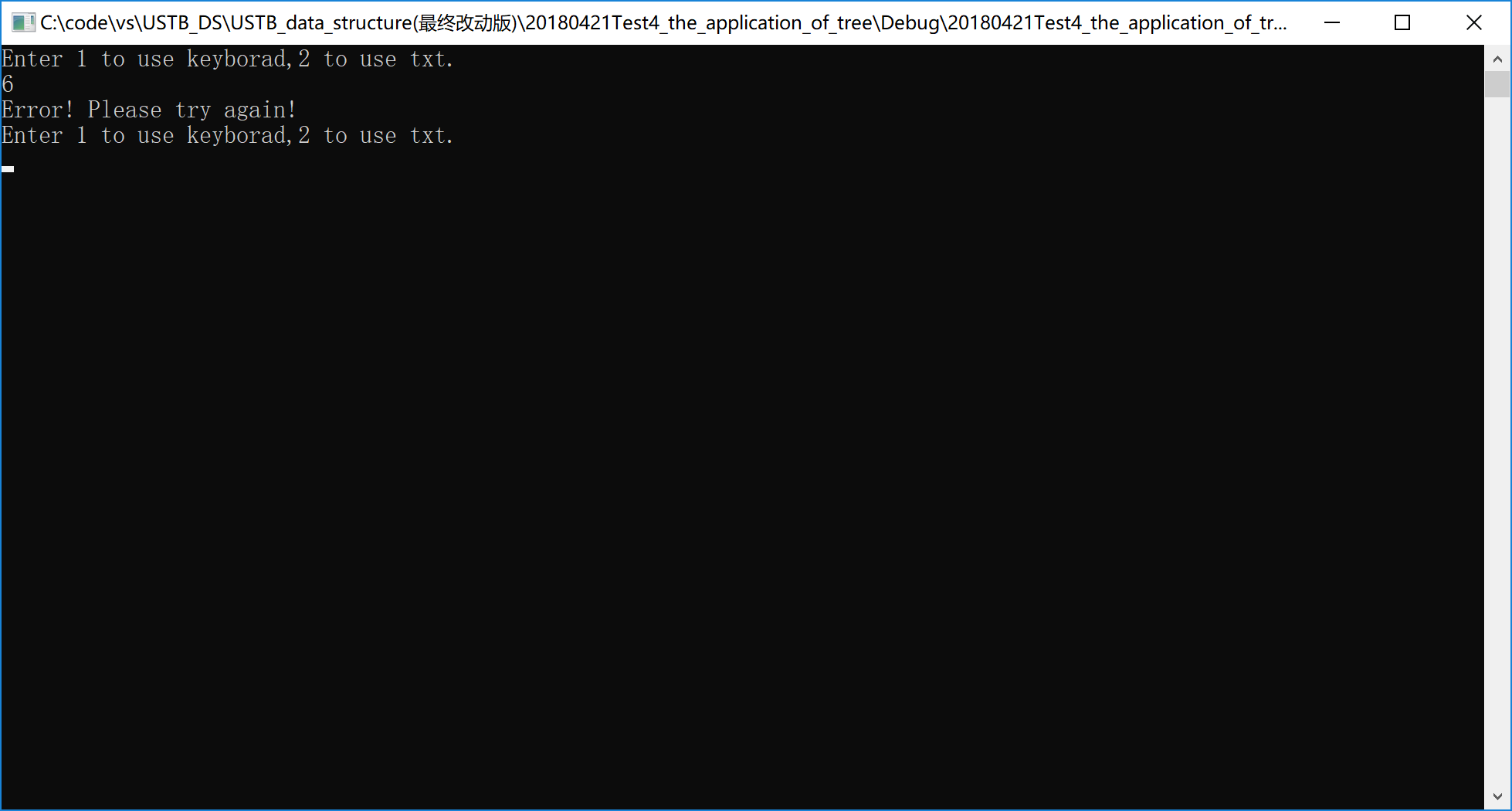


测试txt输入：





测试错误输入:



**算法的有效性分析：**

时间空间复杂度：时间复杂度为O(a+b)，a为字符数，b为单词数。空间复杂度合适

有效性：算法正确，创建和销毁同在，不存在内存泄漏，注释完整，容易阅读，测试完整且正确

不足：代码太长，考验阅读人的耐心