# 二叉树编程练习：---陈鹏宇20204227

## Gbt.h

#ifndef GBT\_H

#define GBT\_H

#include "gbtnode.h"

#include <string>

#include <iostream>

#include <deque>

using namespace std;

template <typename E>

class GBT

{

private:

GBTNode<E>\* root;

inline GBTNode<E>\* create(GBTNode<E>\* R){

E item ; cin >> item; //每次输入一个结点的信息

if(item == '/')

R = nullptr;

else

{

R = new GBTNode<E>;

R->setElement(item);

R->setLeft(create(R->left()));

R->setRight(create(R->right()));

}

return R;

}

inline void clear(GBTNode<E>\* R){

if(R != nullptr)

{

clear(R->left());

clear(R->right());

delete R;

}

}

public:

GBT(){ root = create(root); } //构造

~GBT(){ clear(root); } //析构

GBTNode<E>\* getRoot(){ return this->root; }

inline void preOrder(GBTNode<E>\* R){ //前序遍历

if(R == nullptr)

return;

cout<<R->element()<<" ";

preOrder(R->left());

preOrder(R->right());

}

inline void inOrder(GBTNode<E>\* R){ //中序遍历

if(R == nullptr)

return;

preOrder(R->left());

cout<<R->element()<<" ";

preOrder(R->right());

}

inline void postOrder(GBTNode<E>\* R){ //后序遍历

if(R == nullptr)

return;

preOrder(R->left());

preOrder(R->right());

cout<<R->element()<<" ";

}

inline void levelOrder(){ //层次遍历

deque<GBTNode<E>\*> Q;

if(root == nullptr)

return;

Q.push\_back(root);

while(!Q.empty())

{

GBTNode<E>\* temp = Q.front();

cout<<temp->element()<<" ";

Q.pop\_front();

if(temp->left() != nullptr)

Q.push\_back(temp->left());

if(temp->right() != nullptr)

Q.push\_back(temp->right());

}

}

};

#endif // GBT\_H

## Gbtnode.h

#ifndef GBTNODE\_H

#define GBTNODE\_H

#include "binnode.h"

template <typename E>

class GBTNode //一般二叉树

{

private:

E it;

GBTNode\* lc;

GBTNode\* rc;

public:

GBTNode(){ lc = nullptr ; rc = nullptr; }

GBTNode(E e , GBTNode\* l , GBTNode\* r = nullptr){

it = e;

lc = l;

rc = r;

}

~GBTNode() {}

E& element(){return it;}

void setElement(const E& e){it = e;}

inline GBTNode\* left()const{return lc;}

void setLeft(GBTNode<E>\* b){lc = (GBTNode\*)b;}

inline GBTNode\* right()const{return rc;}

void setRight(GBTNode<E>\* b){rc = (GBTNode\*)b;}

bool isLeaf(){return (lc == nullptr) && (rc == nullptr);}

};

#endif // GBTNODE\_H

## Main.cpp

### 1.将BST树的节点和树的类型定义改写为一般二叉树的节点和树的类型。实现二叉树的构造函数（输入为二叉树的顺序表示法的前序遍历序列，输出为二叉树的二叉链表），并打印出二叉树的前序、中序和后序遍历序列和层次遍历。

#include <iostream>

#include "gbt.h"

#include <stack>

using namespace std;

int main()

{

//INPUT: A B D / G / / / C E / / F / /

//1.将BST树的节点和树的类型定义改写为一般二叉树的节点和树的类型。

// 实现二叉树的构造函数（输入为二叉树的顺序表示法的前序遍历序列，输出为二叉树的二叉链表）

// 并打印出二叉树的前序、中序和后序遍历序列和层次遍历。

GBT<char>\* tree = new GBT<char>();

GBTNode<char>\* root = tree->getRoot();

cout<<"preoder: ";

tree->preOrder(root); // 前序遍历

cout << endl;

cout<<"inorder: ";

tree->inOrder(root); // 中序遍历

cout << endl;

cout<<"postorder: ";

tree->postOrder(root); // 后序遍历

cout << endl;

cout<<"levelorder: ";

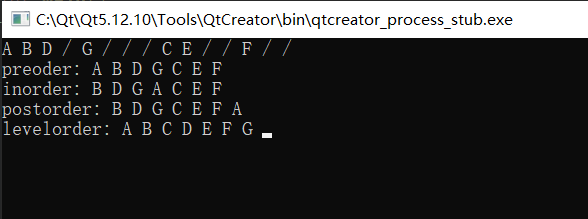
tree->levelOrder(); // 层序遍历

cout << endl;

delete tree;

return 0;

}



### 2.编写一个递归函数search，传入参数为一棵二叉树（不是BST树）的根root和一个值K，如果值K在二叉树中则返回true，否则返回false

#include <iostream>

#include "gbt.h"

#include <stack>

using namespace std;

template <typename E>

bool mySearch(GBTNode<E>\* R , E Key)

{

if(R == nullptr)

return false;

if(R->element() == Key)

return true;

return (mySearch(R->left() , Key) || mySearch(R->right() , Key));

}

Int main(){

//2.编写一个递归函数search，传入参数为一棵二叉树（不是BST树）

//的根root和一个值K，如果值K在二叉树中则返回true，否则返回false

GBT<char>\* tree = new GBT<char>();

char Key ; cin >> Key;

GBTNode<char>\* root = tree->getRoot();

if(!mySearch(root , Key))

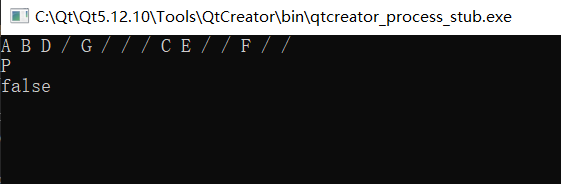
cout<<"false"<<endl;

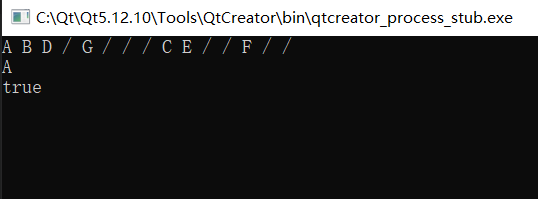
else

cout<<"true"<<endl;

return 0;

}





### 3.编写一个递归函数，求二叉树的高度。

#include <iostream>

#include "gbt.h"

#include <stack>

using namespace std;

template <typename E>

int treeHeight(GBTNode<E>\* R)

{

if(R == nullptr)

return 0;

int m = treeHeight(R->left());

int n = treeHeight(R->right());

return max(m+1,n+1);

}

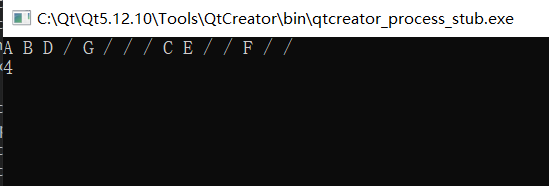
Int main(){

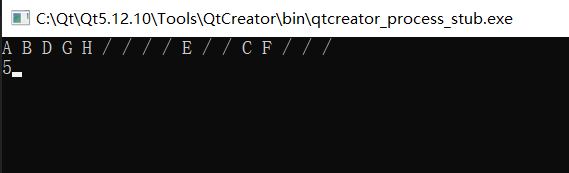
//3.编写一个递归函数，求二叉树的高度。

GBT<char>\* tree = new GBT<char>();

cout<<treeHeight(tree->getRoot());

}





### 4.编写一个递归函数，求二叉树中叶子节点的个数。

#include <iostream>

#include "gbt.h"

#include <stack>

using namespace std;

template <typename E>

int leafNum(GBTNode<E>\* R)

{

if(R == nullptr)

return 0;

if(R->left() == nullptr && R->right() == nullptr)

return 1;

return leafNum(R->left()) + leafNum(R->right());

}

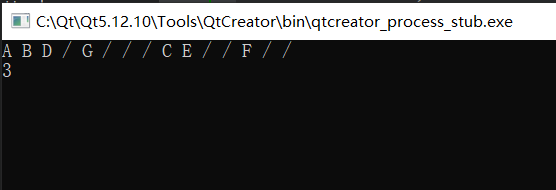
Int main(){

//4.编写一个递归函数，求二叉树中叶子节点的个数

GBT<char>\* tree = new GBT<char>();

cout<<leafNum(tree->getRoot());

}



### 5.编写一个非递归函数，求二叉树的前序遍历。

#include <iostream>

#include "gbt.h"

#include <stack>

using namespace std;

template <typename E>

void newPreOrder(GBTNode<E>\* R)

{

stack<GBTNode<E>\*> S;

S.push(R);

while(!S.empty())

{

GBTNode<E>\* temp = S.top();

S.pop();

cout<<temp->element()<<" ";

if(temp->right() != nullptr)

S.push(temp->right());

if(temp->left() != nullptr)

S.push(temp->left());

}

}

Int main(){

//5.编写一个非递归函数，求二叉树的前序遍历。

GBT<char>\* tree = new GBT<char>();

cout<<"NewPreOrder: ";

newPreOrder(tree->getRoot());

}

