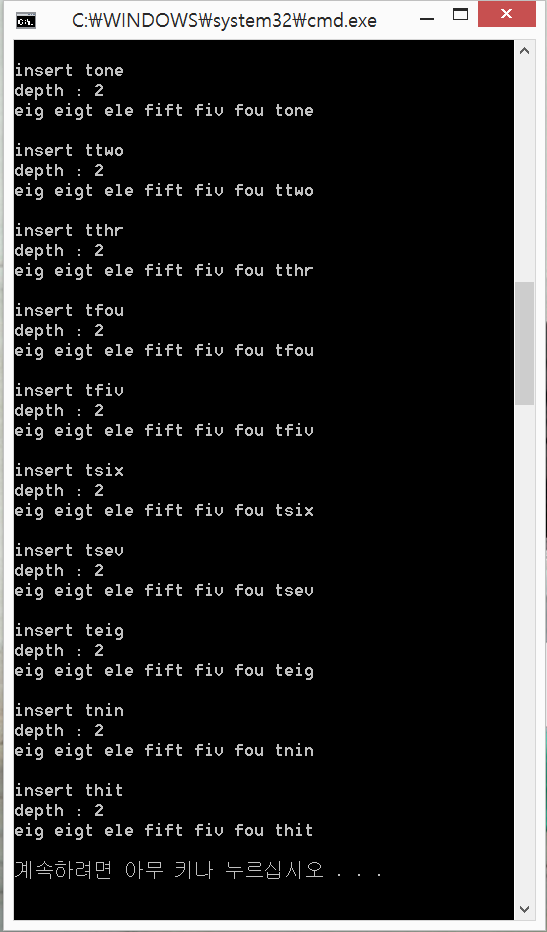
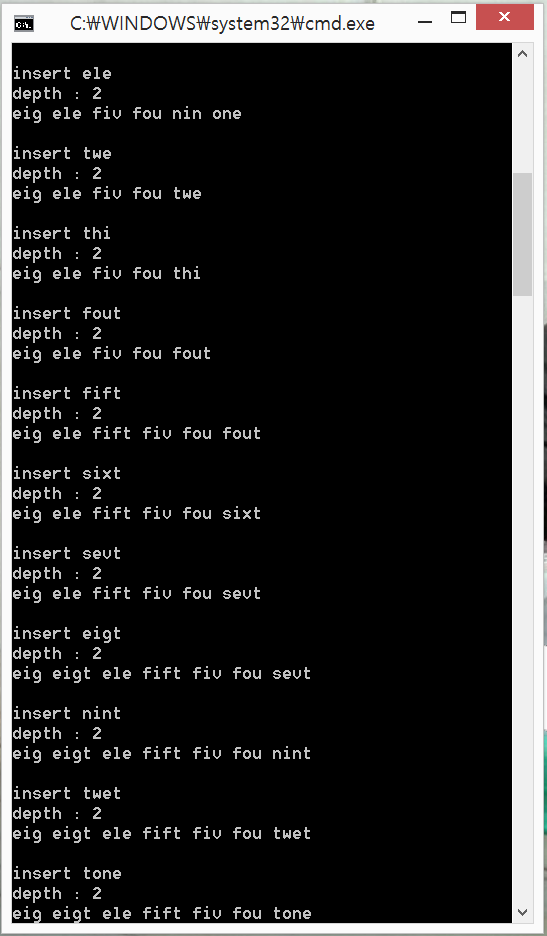
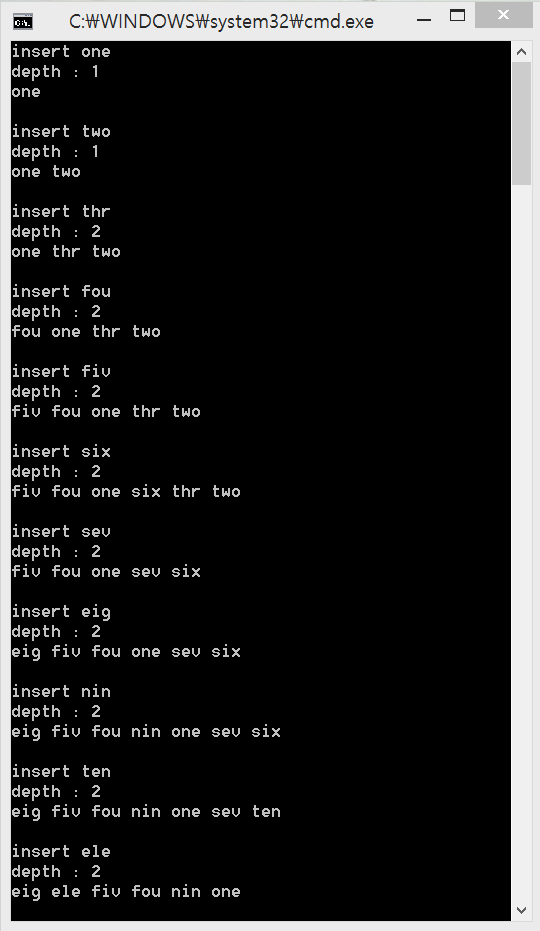
실행결과



Source.cpp

#include "FS.h"

void main(){

Node\* root = new Node;

int i;

for(i=0; i<KEYNUM; i++) {

root->insert(&root, number[i]);

cout << "insert " << number[i] << endl;

cout << "depth : " << root->getDepth() << endl;

root->printTree();

cout << endl << endl;

}

}

FS.h

#include <iostream>

#include <cstring>

#include <string>

#include <algorithm>

#define KEYNUM 30

#define size 3

using namespace std;

string number[KEYNUM] = {

"one",

"two",

"thr",

"fou",

"fiv",

"six",

"sev",

"eig",

"nin",

"ten",

"ele",

"twe",

"thi",

"fout",

"fift",

"sixt",

"sevt",

"eigt",

"nint",

"twet",

"tone",

"ttwo",

"tthr",

"tfou",

"tfiv",

"tsix",

"tsev",

"teig",

"tnin",

"thit"

};

class Node{

private:

string key[size-1];

Node\* childPtr[size];

bool keyIsFull(){

for(int i=0; i<size-1; i++)

if(key[i].empty()) return false;

return true;

}

bool isLeafNode(){

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

if(childPtr[i] !=NULL) return false;

return true;

}

void searchNode(string key, Node\*\* p, Node\*\* q){

if((\*p)->isLeafNode())

return ;

\*q = \*p;

for(int i=0; i<size-1; i++) {

if( this->key[i].compare(key) == 0)

return ;

else if( this->key[i].compare(key) < 0 ) ;

else {

\*p = this->childPtr[i];

return searchNode(key, p,q);

}

}

}

void swapPtr(Node\*\* ptr1, Node\*\* ptr2){

Node\* temp = \*ptr1;

\*ptr1 = \*ptr2;

\*ptr2 = temp;

}

void insertToNode(Node \*\*p, string key){

int i;

for(i=0; i<size && !(\*p)->key[i].empty() && (\*p)->key[i].compare(key)<0; i++) ;

(\*p)->key[i].swap(key);

i++;

Node\* temp = (\*p)->childPtr[i];

(\*p)->childPtr[i+1] = NULL;

for( ; i<size-1; i++){

(\*p)->key[i].swap(key);

swapPtr(&(\*p)->childPtr[i], &temp);

}

(\*p)->childPtr[i] = temp;

}

int getKeyIdx(Node\* node, string key){

for(int i=0; !node->key[i].empty() && i<size-1; i++)

if(node->key[i].compare(key) == 0)

return i;

}

public:

Node() {

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

childPtr[i] = NULL;

}

Node(Node\* node){

for(int i=0; i<size-1; i++)

this->key[i] = node->key[i];

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

this->childPtr[i] = node->childPtr[i];

}

void insertKeyAndSplitNode(Node \*\*root, Node\*\* p, Node\*\* q, string key){

int i;

//pretend key is inserted

string tempKeyArr[size];

Node\* tempNodeArr[size+1];

for(i=0; i<size && (\*p)->key[i].compare(key)<0; i++) {

tempKeyArr[i] = (\*p)->key[i];

tempNodeArr[i] = (\*p)->childPtr[i];

}

tempKeyArr[i] = key;

tempNodeArr[i] = NULL;

for( ; i+1<size; i++) {

tempKeyArr[i+1] = (\*p)->key[i];

tempNodeArr[i+1] = (\*p)->childPtr[i];

}

//large keys are stored in new Node(new node = split)

Node split; //temporary node

int middleIdx = size/2;

string middleKey; middleKey.swap(tempKeyArr[middleIdx]); //split pivot

for(i=0; i<middleIdx; i++) {

(\*p)->key[i] = tempKeyArr[i];

(\*p)->childPtr[i] = tempNodeArr[i];

}

(\*p)->key[middleIdx] = "";

for(i=0; i+middleIdx+1<size-1; i++) {

split.key[i].swap(tempKeyArr[middleIdx+i+1]);

split.childPtr[i] = (\*p)->childPtr[middleIdx+i+1];

(\*p)->key[middleIdx+i+1] = "";

(\*p)->childPtr[middleIdx+i+1] = NULL;

}

split.key[i] = tempKeyArr[middleIdx+i+1];

split.childPtr[i] = tempNodeArr[middleIdx+i+1];

if(\*q==NULL){ //replace root

\*q = new Node();

(\*q)->insert(root, middleKey);

(\*q)->childPtr[0] = \*p;

(\*q)->childPtr[1] = new Node(split);

\*root=\*q;

}

else {//if(!(\*q)->keyIsFull()) {

insertToNode(q,middleKey);

int idx = getKeyIdx(\*q, middleKey);

(\*q)->childPtr[idx+1] = new Node(split);

}

}

void insert(Node\*\* root, string key){

/\*

1. serarch the leaf Node to insert key

2. if node is not full : insert key(ascending)

if node is full : split & create new node

\*/

//1. search

Node\* p = this;

Node \*q = NULL;

searchNode(key, &p, &q);

//2-1 : if node is not full

if( !p->keyIsFull() ){

insertToNode(&p, key);

return;

}

//2-2 : if node is full

else{

insertKeyAndSplitNode(root, &p, &q, key);

}

}

void printTree(){

int i;

for(i=0; i<size-1; i++){

if(this->childPtr[i]!=NULL)

this->childPtr[i]->printTree();

if( !this->key[i].empty() )

cout << this->key[i] << " ";

}

if(this->childPtr[i]!=NULL)

this->childPtr[i]->printTree();

}

int getDepth(){

int Depth[size];

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

Depth[i] = 0;

if(this->isLeafNode())

return 1;

for(int i=0; i<size; i++){

if( this->childPtr[i] !=NULL)

Depth[i] = this->childPtr[i]->getDepth();

}

sort(Depth, &Depth[size-1]);

return Depth[0]+1;

}

};