연결리스트

1. 기본 코드

struct Node{

int value;

Node \*next;

};

Node \*head;

void InitList(){

head = (Node\*)malloc(sizeof(Node));

head->next= NULL;

}

1. 표준 라이브러리

#include <list>

|  |  |
| --- | --- |
| lt.back() | lt의 마지막 원소를 참조한다. |
| p=lt.begin() | p는 lt의 첫 원소를 가리키는 반복자다 |
| lt.empty() | lt가 비었는지 조사한다. |
| p=lt.end() || lt.begin() | p는 lt의 끝을 표식하는 반복자다 |
| q=lt.erase(p) | p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다 |
| lt.front() | lt의 첫 번째 원소를 참조한다. |
| q=lt.insert(p,x) | p가 가리키는 위치에 x 값을 삽입한다. q는 삽입한 원소를 가리키는 반복자다 |
| lt.pop\_back() | Lt.pop\_front() |
| Lt.push\_back(x) | Lt.push\_front(x) |
| Lt.sort() | 오름차순 정렬 |

이진트리

struct Tree\_Node{

int key;

};

struct Tree{

Tree\_Node TreeArray[size];

int cur= 0;

};

void Add\_Tree(Tree \*Target\_Tree, int Item){

Target\_Tree->TreeArray[Target\_Tree->cur].key = Item;

Target\_Tree->cur += 1;

}

void Show\_Right\_Child(Tree \*Target\_Tree, int POSITION){

if(POSITION \* 2 > MAX\_SIZE){

}

}

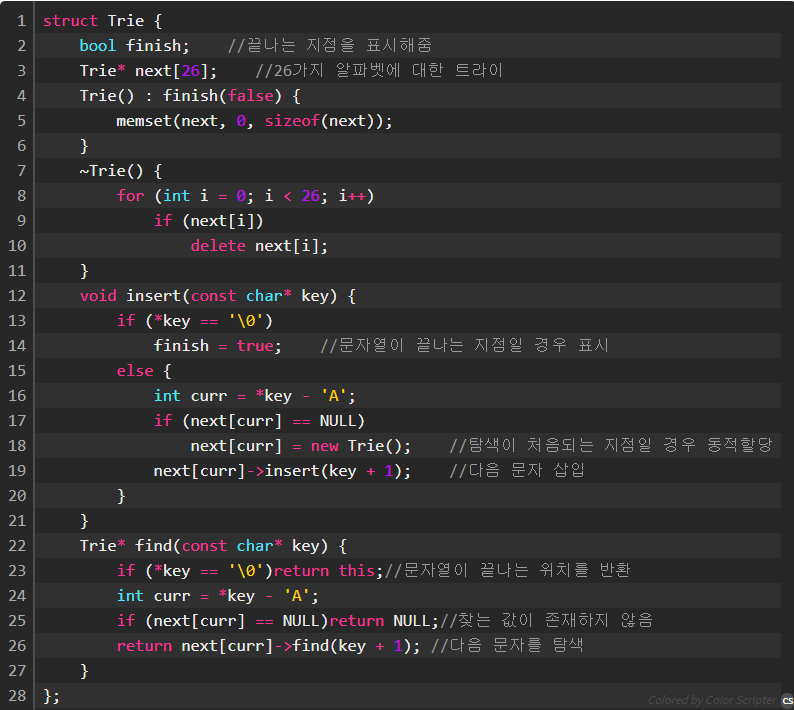
void Show\_Parents(Tree \*Target\_Tree, int Position){

if(Position %2 == 1){

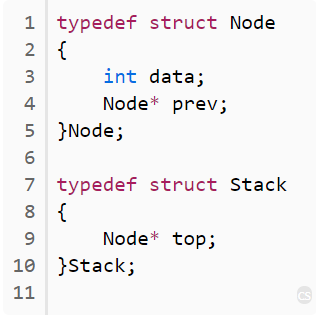
(Position-1)/2;

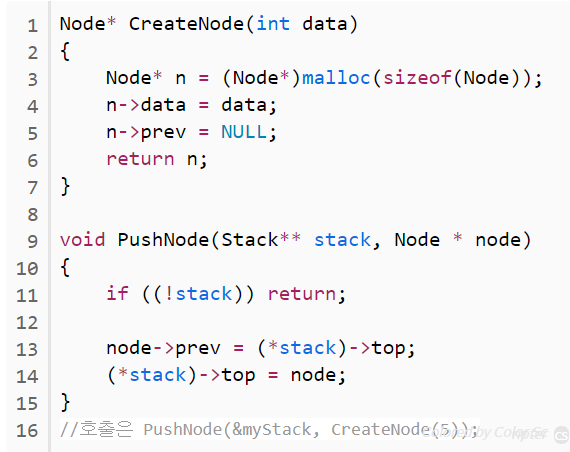
}

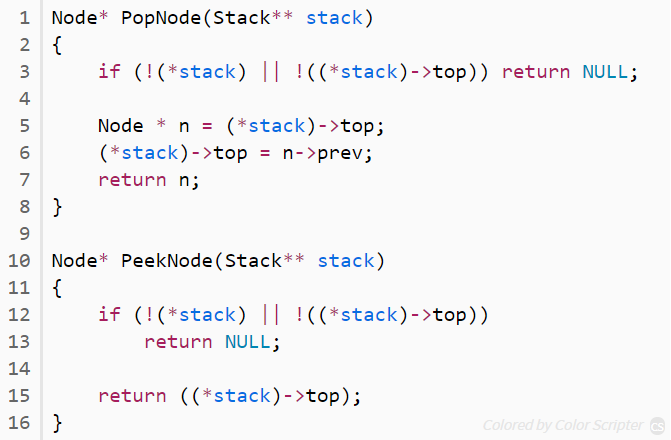
트라이



스택







- stack<int> st : 디폴트 컨테이너 deque<int>를 사용하여 stack 컨테이너 객체를 생성한다.

- stack<int, vector<int>> st : vector 컨테이너를 적용한 정수를 저장하는 stack 컨테이너를 생성한다.

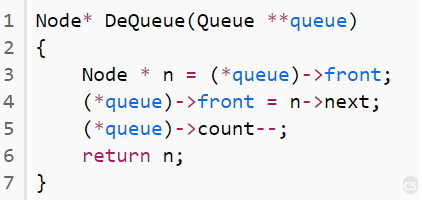
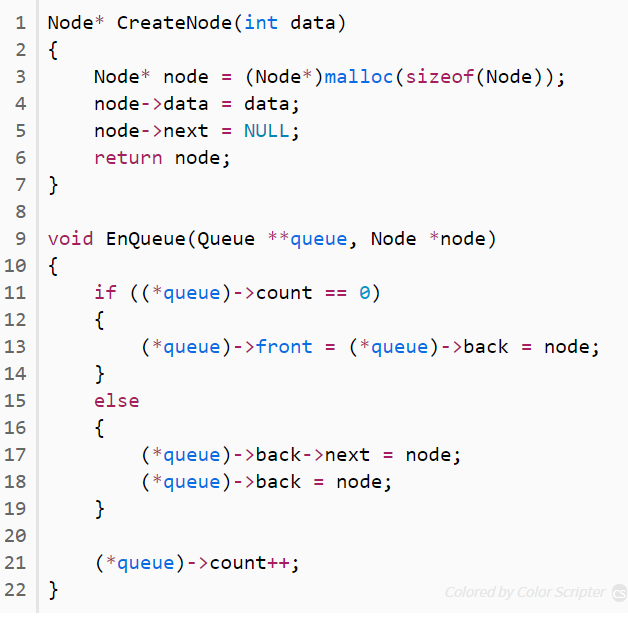
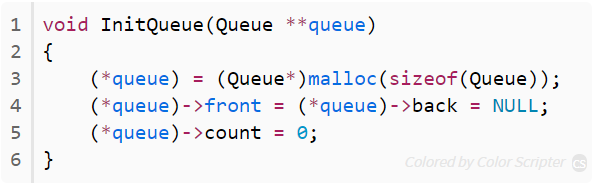
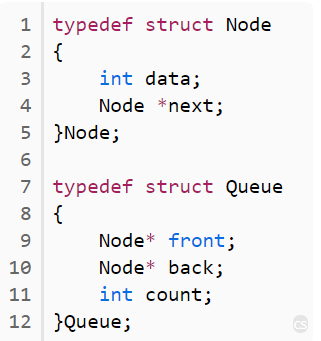
- st.push(x) : 스택에 데이터 x를 입력한다.

- st.pop() : 스택의 데이터를 삭제한다.

- st.top() : 스택의 데이터를 반환한다.

- st.empty() : 스택이 비어있는지 판단한다.

큐



#include<queue>

queue<int> Q  //Q라는 이름의 int 요소들로 구성된 큐 선언

Q.push(값)  //큐 Q에 값을 넣는다. 리턴 값이 없다.

Q.pop()  //큐 Q의 front를 삭제한다. 리턴 값이 없다.

Q.front()  //큐 Q의 front를 리턴한다. front는 삭제되지 않는다. (peek기능)

Q.back()  //큐 Q의 back를 리턴한다. back는 삭제되지 않는다. (peek기능)

Q.size()  //큐 Q의 크기(구성 요소 갯수)를 리턴한다.

Q.empty()  //큐 Q가 비어있으면(요소가 없으면) 를 1(True)리턴하고 비어있지 않으면 0(False)를 리턴한다.

벡터

#include <vector>

vector v;

v.at(i) - i번째 원소 참조

v.front( ) || v.back() – v의 마지막 원소를 참조

p= v.begin( ) || v.end( ) – v의 첫 원소를 가리키는 반복자

v.empty()

v.pop\_back( ) || v.push\_back( )

sort(v.begin( ), v.end( )); // 오름차순 정렬

sort(v.begin( ), v.end( ), greater<int>( )); // 내림차순 정렬

맵

map<int, string> m;

m.begin( ) || m.end( )

m.find( id ) 반복자

m.empty( ) || m.size( ) || m[1]= “hello” m[2]= “google”

해시 테이블

<http://huiyu.tistory.com/entry/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0-%EC%B2%B4%EC%9D%B4%EB%8B%9D-%ED%95%B4%EC%8B%9C-%ED%85%8C%EC%9D%B4%EB%B8%94Chaining-Hash-Table>

Chaining 방식은 Linked List를 이용하는 방법으로 저장하려는 해시테이블에 이미 같은 키값의 데이터가 있다면,  
노드를 추가하여 다음 노드를 가르키는 방식으로 구현하는 것입니다.

hashTable[1] :(1)->(11)->(21)->(31)->(101)  
hashTable[2] :(2)->(12)->(42)->(62)  
...  
hashTable[9] :(9)->(19)->(99)

