**Test1**

#include <iostream>

#include <random> // C++ 11에서 추가

using namespace std; // C++11 이전: C 스타일 난수 생성(srand와 rand 함수)

#define arrMAXSIZE 15

template <typename E> void HeapSort(E\* pArr, const int num);

template <typename E> void percolateDown(E\* pArr, const int root, const int num);

template <typename E> void SWAP(E\* pa, E\* pb);

template <typename E> void PRINT(E\* pArr, const int num);

int main(void)

{

int arr[arrMAXSIZE] = { 0 };

// C++ 스타일 난수 생성

random\_device rd; // 시드 설정: random\_device 생성

mt19937 gen(rd()); // 난수 생성 엔진(mt19937) 초기화

uniform\_int\_distribution<int> dis(0, 99); // 균등 분포 정의: 범위 지정

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

\*(arr + i) = dis(gen);

cout << "정렬 전: "; PRINT(arr, arrMAXSIZE);

HeapSort(arr, arrMAXSIZE);

cout << "정렬 후: "; PRINT(arr, arrMAXSIZE);

return 0;

}

template <typename E>

void HeapSort(E\* pArr, const int num) {

}

template <typename E>

void percolateDown(E\* pArr, const int root, const int end) {

}

template <typename E> void SWAP(E\* pa, E\* pb) {

E temp;

temp = \*pa;

\*pa = \*pb;

\*pb = temp;

}

template <typename E> void PRINT(E\* pArr, const int num) {

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

cout << pArr[i] << " ";

cout << endl;

}

**Test2**

#include <iostream>

#include "BinarySearchTree.h"

using namespace std;

void searchBST(BinarySearchTree\* bst);

void insertBST(BinarySearchTree\* bst);

void deleteBST(BinarySearchTree\* bst);

int main(void)

{

int num;

BinarySearchTree\* bst = new BinarySearchTree();

while (true) {

system("cls");

cout << "\n ### 이진 탐색 트리 ### \n" << endl;

cout << "1) 데이터 삽입" << endl;

cout << "2) 데이터 검색" << endl;

cout << "3) 데이터 삭제" << endl;

cout << "4) 전체 출력" << endl;

cout << "5) 프로그램 종료" << endl;

cout << "\n메뉴 선택: ";

cin >> num;

switch (num) {

case 1: insertBST(bst); break;

case 2: searchBST(bst); break;

case 3: deleteBST(bst); break;

case 4: bst->printBSTAll(bst->getRoot()); break;

case 5: cout << "프로그램 종료!!!\n" << endl; return 0;

default: cout << "메뉴를 잘못 선택하셨습니다." << endl;

}

system("pause");

}

return 0;

}

// 이진 탐색 트리(BST): 데이터 검색

void searchBST(BinarySearchTree\* bst) {

int num;

DNode\* temp = nullptr;

while (true) {

cout << "\n찾을 임의의 정수 값을 입력하세요(종료: 0): ";

cin >> num;

if (num == 0)

break;

// 데이터검색

temp = bst->search(num);

if (temp) cout << temp->\_\_data << " 키를 찾았습니다!!!" << endl;

else cout << "키를 찾지 못했습니다." << endl;

}

}

// 이진 탐색 트리(BST): 데이터 입력

void insertBST(BinarySearchTree\* bst) {

int num;

while (true) {

cout << "임의의 정수 입력(종료: 0): ";

cin >> num;

if (num == 0)

break;

// 데이터 삽입

bst->insert(num);

}

}

// 이진 탐색 트리(BST): 데이터 삭제

void deleteBST(BinarySearchTree\* bst) {

int num;

DNode\* temp = nullptr;

while (true) {

cout << "\n삭제할 임의의 정수 값을 입력하세요(종료: 0): ";

cin >> num;

if (num == 0)

break;

// 데이터 삭제

bst->remove(num);

}

}

**Test3**

#include <iostream>

#include <stack>

using namespace std;

// GNode class

class GNode {

private:

int \_\_vertex; // 정점

int \_\_weight; // 가중치

GNode\* \_\_link;

friend class GraphType;

public:

GNode(int vertex, int weight);

};

// 그래프 노드(C): 그래프 노드 생성

GNode::GNode(int vertex = 0, int weight = 0)

: \_\_vertex(vertex), \_\_weight(weight), \_\_link(nullptr) {}

// GraphType class

class GraphType {

private:

int \_\_vertex; // 정점의 개수

GNode\*\* \_\_adjSList; // 인접 리스트

public:

GraphType(int vertex);

~GraphType(void);

void insertEdge(int vertex1, int vertex2, int weight);

void DFSAdjSList(int vertex);

void printAdjSList(void) const;

};

// GraphType: 생성자(소멸자)와 메소드 정의

GraphType::GraphType(int vertex) : \_\_vertex(vertex) {

}

// graphDestroy : 그래프 삭제

GraphType::~GraphType(void) {

}

// insertEdge : 간선 추가

void GraphType::insertEdge(int row, int col, int weight) {

}

// 그래프 순회: 깊이 우선 탐색(DFS)

void GraphType::DFSAdjSList(int vertex) {

}

// printAdjMatrix : 그래프 전체 출력

void GraphType::printAdjSList(void) const {

char ch;

for (int i = 0; i < \_\_vertex; i++) {

ch = i + 65;

cout << "\t정점 " << ch << "의 인접 리스트";

GNode\* tNode = \_\_adjSList[i];

while (tNode) {

cout.width(3);

ch = tNode->\_\_vertex + 65;

cout << ch << " ->>";

tNode = tNode->\_\_link;

}

cout << " NULL" << endl;

}

}

int main(void)

{

// G9 : 무향 그래프

GraphType G9 = GraphType(7);

// 정점: A(0)

G9.insertEdge(0, 1, 0); // A(0) - B(1)

G9.insertEdge(0, 2, 0); // A(0) - C(2)

// 정점: B(1)

G9.insertEdge(1, 0, 0); // B(1) - A(0)

G9.insertEdge(1, 3, 0); // B(1) - D(3)

G9.insertEdge(1, 4, 0); // B(1) - E(4)

// 정점: C(2)

G9.insertEdge(2, 0, 0); // C(2) - A(0)

G9.insertEdge(2, 4, 0); // C(2) - E(4)

// 정점: D(3)

G9.insertEdge(3, 1, 0); // D(3) - B(6)

G9.insertEdge(3, 6, 0); // D(3) - G(6)

// 정점: E(4)

G9.insertEdge(4, 1, 0); // E(4) - B(1)

G9.insertEdge(4, 2, 0); // E(4) - C(2)

G9.insertEdge(4, 6, 0); // E(4) - G(6)

// 정점: F(5)

G9.insertEdge(5, 6, 0); // F(5) - G(6)

// 정점: G(6)

G9.insertEdge(6, 5, 0); // G(6) - F(5)

G9.insertEdge(6, 4, 0); // G(6) - E(4)

G9.insertEdge(6, 3, 0); // G(6) - D(3)

cout << "\n##### 그래프(G9): 인접 리스트 #####\n" << endl;

G9.printAdjSList();

printf("\n##### 그래프(G9): 깊이 우선 탐색(DFS) #####\n\n");

G9.DFSAdjSList(0);

return 0;

}

**BinarySearchTree**

#pragma once

#include <iostream>

#include <queue>

using namespace std;

class DNode {

public:

int \_\_data;

DNode\* \_\_Llink;

DNode\* \_\_Rlink;

DNode(const int& data);

};

// 새로운 노드(data, link) 생성

DNode::DNode(const int& data) :

\_\_data(data), \_\_Llink(nullptr), \_\_Rlink(nullptr) {}

class BinarySearchTree {

private:

DNode\* \_\_root;

DNode\* \_\_search(DNode\* root, const int& data) {

}

void \_\_insert(DNode\* root, const int& data) {

}

public:

DNode\* getRoot(void) const { return \_\_root; }

DNode\* search(const int& data);

void insert(const int& data);

void remove(const int& data);

void printBSTAll(DNode\* root) const;

};

// 전체 노드 출력 -- 중위 순회

void BinarySearchTree::printBSTAll(DNode\* root) const {

if (root) {

printBSTAll(root->\_\_Llink);

cout << root->\_\_data << " ";

printBSTAll(root->\_\_Rlink);

}

}

// 데이터 검색

DNode\* BinarySearchTree::search(const int& data) {

}

// 데이터 삽입

void BinarySearchTree::insert(const int& data) {

}

// 데이터 삭제

void BinarySearchTree::remove(const int& data) {

}