2023 November 20

**자료구조와 알고리즘 과제제출**

실습보고서 3

로고이(가) 표시된 사진

자동 생성된 설명

**담당교수 : 서두옥교수님**

**학과 : 자동차it융합과**

**학번 : 20215229**

**이름 : 이채훈**

**제출일 : 2023년11월20일**

**1. 괄호의 값: (BOJ) BaekJoon 2504**

**소스코드**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

//( : -1, ) : -2, [ : -3, ]: -4

int bracket(string str) {

stack <int> s;

int len = str.length();

int temp = 0;

int result = 0;

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

//열린 괄호인 경우

if (str[i] == '(') s.push(-1);

//닫힌 대괄호인 경우

else if (str[i] == '[') s.push(-3);

//예외1 : 닫힌 괄호가 더 많은 경우

else if (s.empty())

return 0;

else {

//스택의 맨위가 열린 괄호이고, 현재 괄호가 닫힌 괄호인 경우

if (s.top() == -1 && str[i] == ')') {

s.pop();

s.push(2);

}

//스택의 맨위가 열린 대괄호이고, 현재 괄호가 닫힌 대괄호인 경우

else if (s.top() == -3 && str[i] == ']') {

s.pop();

s.push(3);

}

//s.top()이 숫자인 경우, 예외3 : 괄호의 짝이 맞지 않는 경우

else {

temp = 0;

//스택이 비거나, 열린 괄호를 만날 때까지 값을 더함

while (s.top() != -1 && s.top() != -3) {

temp += s.top();

s.pop();

// 예외1 : 닫힌 괄호가 더 많은 경우

if (s.empty())

return 0;

}

//스택의 맨 위가 열린 괄호이고, 현재 괄호가 닫힌 괄호인 경우

if (s.top() == -1 && str[i] == ')') {

temp \*= 2;

s.pop();

}

//스택의 맨 위가 열린 대괄호이고, 현재 괄호가 닫힌 대괄호인 경우

else if (s.top() == -3 && str[i] == ']') {

temp \*= 3;

s.pop();

}

// 예외3 : 괄호의 짝이 맞지 않는 경우

else

return 0;

s.push(temp);

}

}

}

//스택에 남아 있는 값들을 더함

while (!s.empty()) {

// 예외 2 : 열린괄호가 더 많은 경우

if (s.top() == -1 || s.top() == -3)

return 0;

result += s.top();

s.pop();

}

return result;

}

int main() {

string st;

cout << " 20215229 이채훈 " << endl;

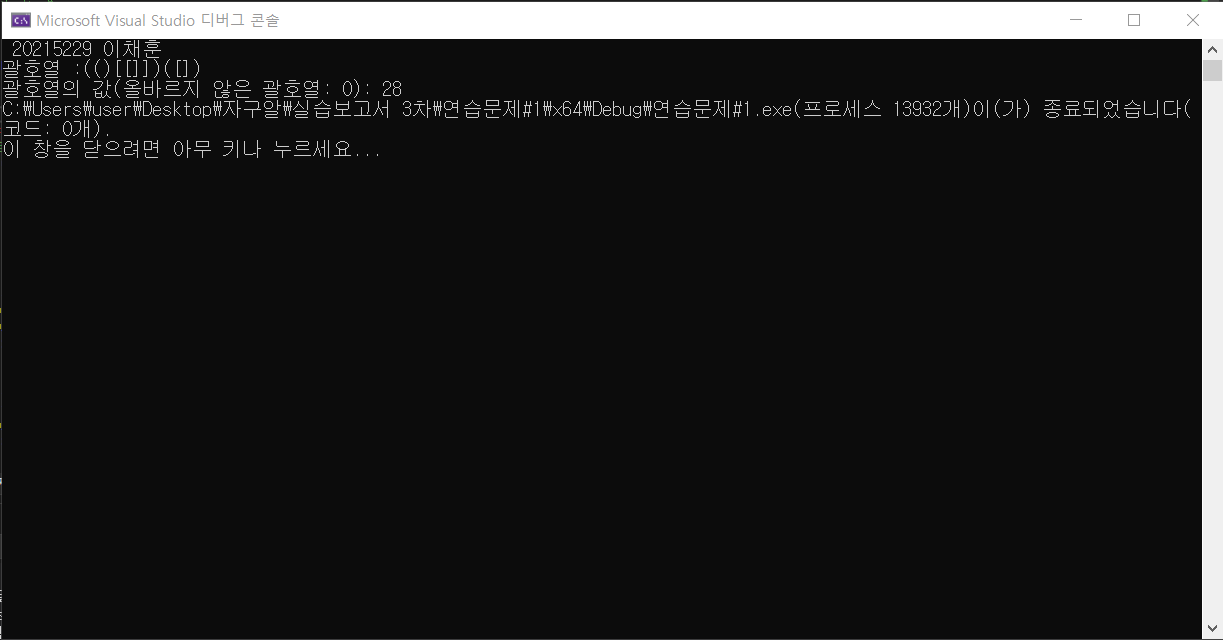
cout << "괄호열 :"; cin >> st;

cout << "괄호열의 값(올바르지 않은 괄호열: 0): "<<bracket(st);

}

**실행결과**

**# 올바른 괄호열**

****

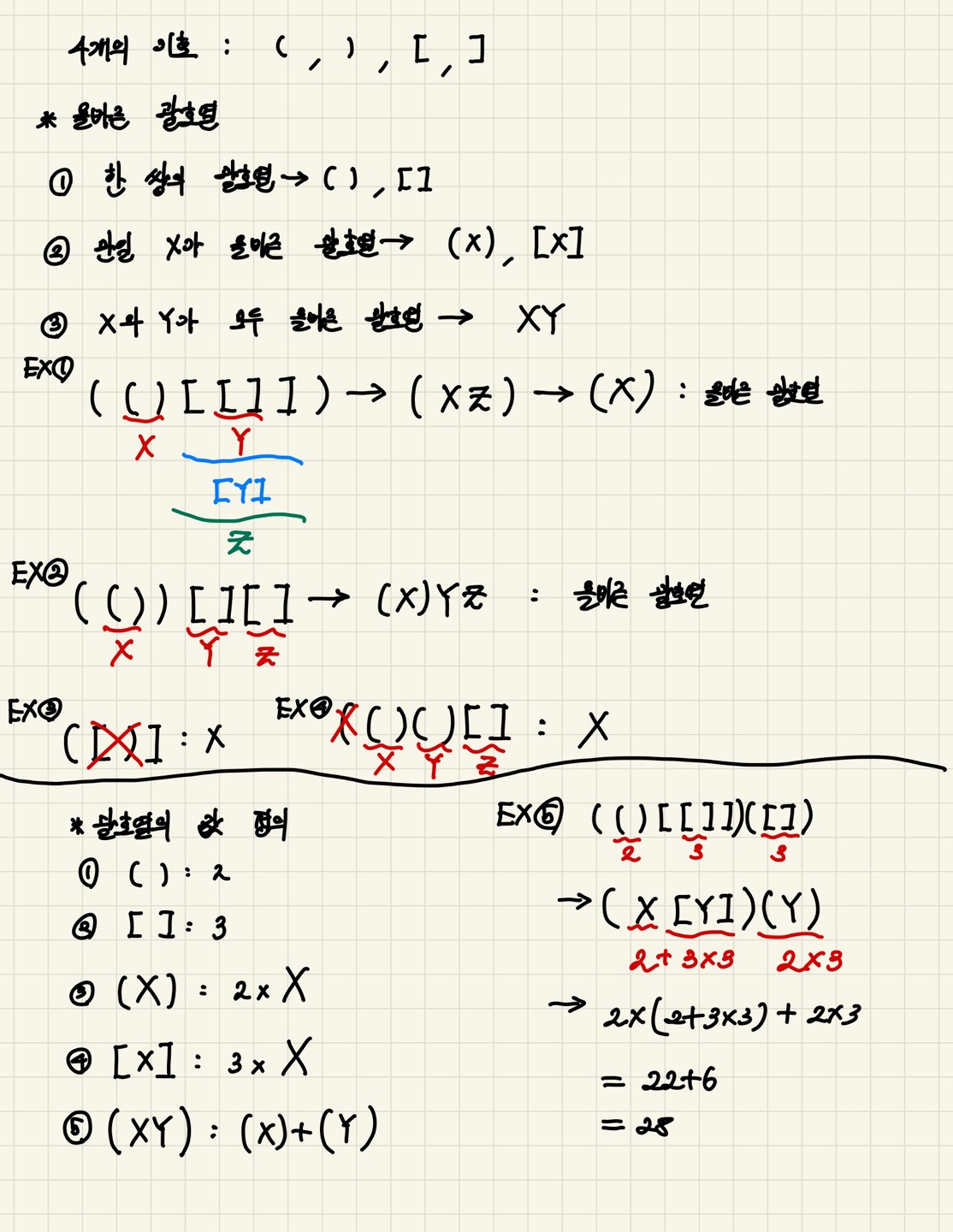
**# 올바르지 않은 괄호열 : 0**

**텍스트, 스크린샷, 컴퓨터, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**# 문제해석**

****

**# 문제해법**

EX5 : (()[[]])([])

(

( (

( ( ) -> ( 2

( 2 [

( 2 [ [

( 2 [ [ ] -> ( 2 [ 3

( 2 [ 3 ] -> ( 2 9

( 2 9 ) -> (2 + 9) \* 2 = 22

22 (

22 ( [

22 ( [ ] -> 22 ( 3

22 ( 3 ) -> 22 6

ans= 22 + 6 = 28

0. temp : 괄호열의 부분값을 임시로 저장하는 변수, result : 최종 괄호열의 값

1. 주어진 문자열을 for문으로 순회하면서 열린 괄호일 경우 스택에 해당 괄호의 값을 넣음 (-1 또는 -3).

2. 닫힌 괄호일 경우 스택의 맨 위에 있는 값과 짝이 맞는지 확인하고, 값을 스택에 넣음 (2 또는 3).

3. **스택이 비어있는데 닫힌 괄호가 나오는 경우 예외 처리** (닫힌 괄호가 더 많은 경우).

4. **스택의 맨 위에 있는 값이 숫자인 경우, 괄호의 짝이 맞지 않는 경우 예외 처리.**

5. 최종적으로 스택에 남아 있는 값들을 더하여 괄호열의 전체 값 계산.

**<예외 : 괄호의 짝이 맞지 않는 경우>**

: 예외 처리가 올바르게 되지 않을 경우, 코드가 이상하게 변하여서, 여러가지 테스트를 통해서 예외처리에 대한 경우의 수와 예외처리가 필요해서 예외처리에 집중하였다.

1. 닫힌 괄호왔는데 스택이 비어있는 경우(닫힌 괄호가 더 많은 경우)

2. 반복문이 끝났는데 스택이 비어있지 않는 경우(열린 괄호가 많다)

3. 닫힌괄호와 열린괄호의 짝이 맞지 않는 경우

**2. 스택구현: 순차 자료구조**

**소스코드**

#include <iostream>

#define stackMAXSIZE 10

using namespace std;

inline void error(const char\* message);

// 배열 스택: arrayStack

template <typename E>

class arrayStack {

private:

E stack[stackMAXSIZE];

int top;

public:

arrayStack();

~arrayStack();

void push(const E& e);

E pop(void);

E peek(void) const;

int size(void) const;

bool Empty(void) const;

bool Full(void) const;

void printStack(void) const;

};

// 생성자: 빈 스택 생성

template <typename E>

arrayStack<E>::arrayStack() : top(-1) {}

// 소멸자

template <typename E>

arrayStack<E>::~arrayStack() {}

// PUSH : 스택에 데이터 삽입

template <typename E>

void arrayStack<E>::push(const E& e) {

if (Full())

error("스택 포화 에러");

stack[++top] = e;

}

// POP : 스택에서 데이터 삭제

template <typename E>

E arrayStack<E>::pop(void) {

if (Empty())

error("스택 공백 에러");

return stack[top--];

}

// PEEK : 스택 맨 위의 원소 확인

template <typename E>

E arrayStack<E>::peek(void) const {

if (Empty())

error("스택 공백 에러");

return stack[top];

}

// SIZE : 스택의 크기 값 반환

template <typename E>

int arrayStack<E>::size(void) const {

return top + 1;

}

// 스택의 공백 상태 여부 판단

template <typename E>

bool arrayStack<E>::Empty(void) const {

return top == -1;

}

// 스택의 포화 상태 여부 판단

template <typename E>

bool arrayStack<E>::Full(void) const {

return top == stackMAXSIZE - 1;

}

// 스택의 전체 원소 출력

template <typename E>

void arrayStack<E>::printStack(void) const {

cout << "\nSTACK [";

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

cout.width(3);

cout << stack[i];

}

cout << " ]" << endl;

}

inline void error(const char\* message) {

cout << message;

exit(100);

}

int main(void)

{

int num, choice;

arrayStack<int> s = arrayStack<int>();// 빈 스택 생성

while (true) {

system("cls");

cout << " 20215229 이채훈 " << endl;

cout << "\n ### 스택 구현: 1차원 배열 ###\n" << endl;

cout << "1) 데이터 삽입: PUSH" << endl;

cout << "2) 데이터 삭제: POP" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

s.push(num);

break;

case 2: cout << "삭제 된 데이터: " << s.pop() << endl;

break;

case 3: s.printStack();

cout << "스택 크기: " << s.size() << endl;

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

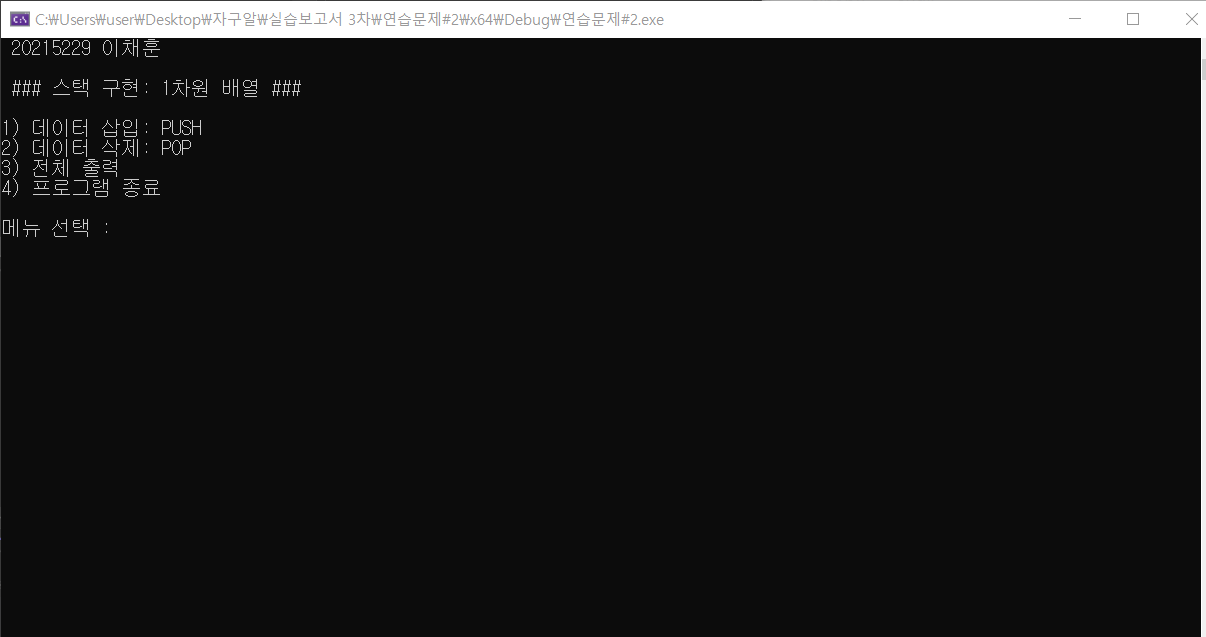
}

return 0;

}

**실행결과**

**# main 함수 출력화면**

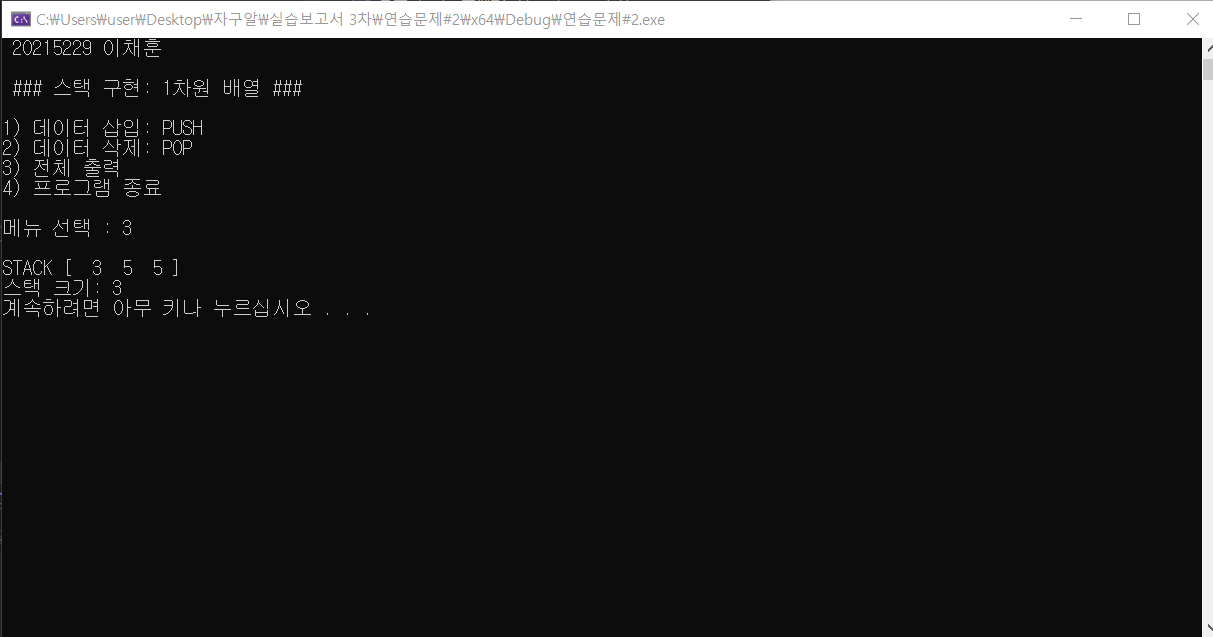
****

**# 스택 공백 에러**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**# printstack, size 출력화면**

****

**연구조사**

**# 에러 처리 함수 정의**

: `error` 함수는 주어진 메시지를 출력하고 프로그램을 종료시킴.

inline void error(const char\* message) {

cout << message;

exit(100);

}

* error("스택 포화 에러");
* error("스택 공백 에러");

**# 스택 기본 동작 함수들**

`push`: 스택에 데이터를 삽입하는 함수입

`pop`: 스택에서 데이터를 삭제하고 해당 데이터를 반환하는 함수

`peek`: 스택의 맨 위에 있는 원소를 확인하는 함수

`size`: 스택의 현재 크기를 반환하는 함수

`Empty`: 스택이 비어있는지 확인하는 함수

`Full`: 스택이 꽉 차 있는지 확인하는 함수

`printStack`: 스택의 전체 원소를 출력하는 함수

**3. 원형 큐 구현: 순차 자료구조**

**소스코드**

#include <iostream>

using namespace std;

inline void error(const char\* message);

#define queueMAXSIZE 10

template <typename E>

class CircularQueue {

private:

E queue[queueMAXSIZE];

int \_\_front, \_\_rear, \_\_count;

public:

//생성자 & 소멸자

CircularQueue();

~CircularQueue();

//push, pop,front, back

void push(const E& e);

E pop(void);

E front(void) const;

E back(void) const;

//size, empty, full

int size(void) const;

bool empty(void) const;

bool full(void) const;

//printQueue

void printQueue(void) const;

};

template<typename E>

CircularQueue<E>::CircularQueue()

: \_\_front(-1), \_\_rear(-1), \_\_count(0) {}

template<typename E>

CircularQueue<E>::~CircularQueue() {}

template<typename E>

void CircularQueue<E>::push(const E& e) {

if (full())

error("원형 큐 포화 에러");

else {

\_\_rear = (\_\_rear + 1) % queueMAXSIZE;

queue[\_\_rear] = e;

\_\_count++;

}

}

template<typename E>

E CircularQueue<E>::pop() {

if (empty())

error("원형 큐 공백 에러");

else {

\_\_count--;

return queue[++\_\_front];

}

}

template<typename E>

E CircularQueue<E>::front() const {

if (empty())

error("원형 큐 공백 에러");

else

return queue[(\_\_front + 1) % queueMAXSIZE];

}

template<typename E>

E CircularQueue<E>::back() const {

if (empty())

error("원형 큐 공백 에러");

else

return queue[\_\_rear];

}

template<typename E>

int CircularQueue<E>::size() const {

return \_\_count;

}

template<typename E>

bool CircularQueue<E>::empty() const {

return \_\_front == \_\_rear;

}

template<typename E>

bool CircularQueue<E>::full() const {

return \_\_count == queueMAXSIZE;

}

template<typename E>

void CircularQueue<E>::printQueue() const {

if (empty())

error("원형 큐 공백 에러");

else {

if (\_\_rear > \_\_front) {

for (int i = \_\_front + 1; i <= \_\_rear; i++) {

cout.width(3);

cout << queue[i];

}

}

else if (\_\_rear < \_\_front) {

for (int i = \_\_front + 1; i < queueMAXSIZE; i++) {

cout.width(3);

cout << queue[i];

}

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

cout.width(3);

cout << queue[i];

}

}

cout << endl;

}

}

inline void error(const char\* message) {

cout << message;

exit(100);

}

int main(void)

{

int num, choice;

CircularQueue<int> CQ= CircularQueue<int>();// 빈 큐 생성

while (true) {

system("cls");

cout << " 20215229 이채훈 " << endl;

cout << "\n ### 큐 구현: 1차원 배열 ### \n" << endl;

cout << "1) 데이터 삽입(enQueue): " << endl;

cout << "2) 데이터 삭제(deQueue): " << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

CQ.push(num);

break;

case 2: cout << "삭제 된 데이터: " << CQ.pop() << endl;

break;

case 3: CQ.printQueue();

cout << "Front: " << CQ.front() << ", Rear: " << CQ.back() << ", Size: "<<CQ.size()<<endl;

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

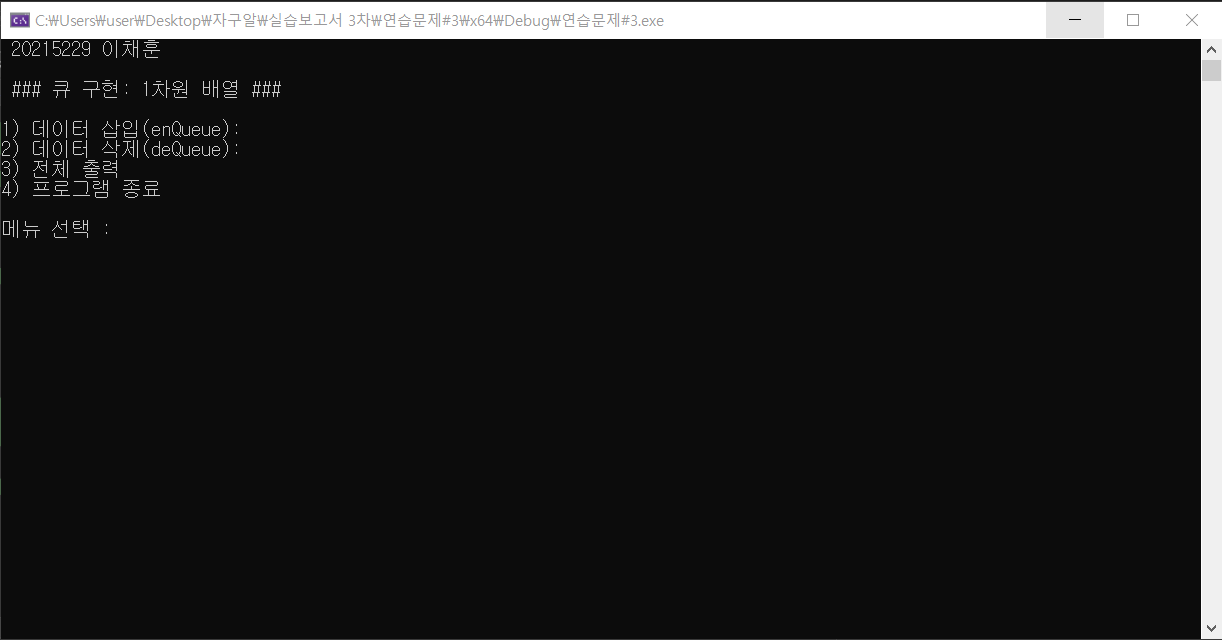
}

return 0;

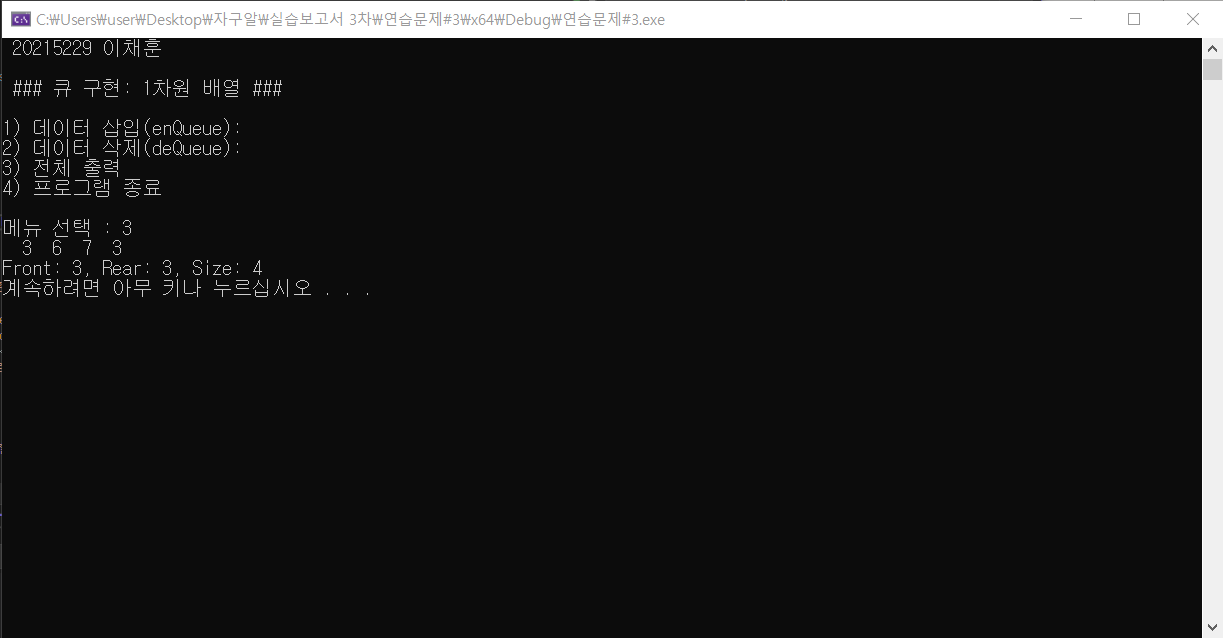
}

**실행결과**

**# main함수 화면 출력**

****

**#원형 큐 공백 에러**

****

**# printQueue, front, back, size 출력화면**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**# Circular Queue**

생성자&소멸자

큐함수- push(enqueue), pop(dequeue)

탐색-empty(), full()

출력-printStack(), front(), back(), size()

**1. empty**

큐가 비어있는지 확인한다. front와 rear가 같으면 큐가 비었다고 판단한다.

**2. full**

큐가 꽉 찼는지 확인한다. 선형 큐의 경우, rear+1의 값이 queueMaXSIZE와 같은지 확인하면 되지만,

원형 큐는 같은 방법으로 할 경우, front = rear인 경우가 생겨, empty와 구분할 수 없다.

따라서, 별도의 count 변수를 유지하여 데이터의 갯수를 세고, count == queueMaXSIZE와 같은지 확인해야한다.

**3. printQueue**

큐의 모든 데이터를 확인한다.

rear > front인 경우는 선형 큐와 같은 상황이므로 front+1부터 rear까지 출력하면 되지만,

rear < front인 경우는 queue의 마지막 칸까지 채워진 상태에서, 비어있는 앞칸에 데이터를 삽입하는 경우이므로 두 부분을 나누어 출력해준다.

**4. push(enQueue)**

rear에 data를 입력한다.

(rear+1)%queueMAXSIZE의 의미 : rear값을 반드시 0~9 사이값으로 만든다.

이는 rear가 queueMAXSIZE-1 의 인덱스에 도달할 경우, rear를 다시 0부터 시작하도록 해주며, 앞칸부터 다시 삽입할 수 있도록 해준다

-> 꽉 차있는지 확인

-> 꽉 차지 않은 경우 rear = (rear+1)%queueMAXSIZE로 삽입할 자리 선정

-> 이 후 삽입 실행

-> count 갯수 증가

**5. pop(deQueue)**

front에서 data를 삭제한다.

-> 비어있는지 확인

-> 비어있지 않은 경우, count 갯수 감소 후 , 현재 front의 다음칸을 반환

**4. 이진 트리의 순회: 비재귀적 용법**

**소스코드**

**# LinkedBtree.cpp**

#include <iostream>

#include <string>

#include "LinkedBtree.h"// class DNode

using namespace std;

int main(void)

{

string postfix;

cout << "20215229 이채훈" << endl;

cout << "\n트리를 구성할 후위 수식: ";

getline(cin, postfix);

// 이진 트리 생성 및 구성

LinkedBTree<char> BT;

DNode<char>\* root = BT.makeLinkedBTree(postfix);

// 깊이 우선 순회: 전위.중위.후위 순회

cout << "\nPreorder : "; BT.Preorder(root); // 전위 순회

cout << "\nInorder : "; BT.Inorder(root); // 중위 순회

cout << "\nPostorder : "; BT.Postorder(root); // 후위 순회

// 너비 우선 순회

cout << "\nLevelorder : "; BT.Levelorder(root);

return 0;

}

**# LinkedBtree.h**

#include <iostream>

#include <string>

#include <stack>

#include <queue>

#include "LinkedQueue.h"

using namespace std;

// 연산자 여부를 판단

int isOperator(int op) {

return op == '+' || op == '-' || op == '\*' || op == '/';

}

template <typename E>

class LinkedBTree;

// class DNode

template <typename E>

class DNode {

public:

DNode(const E& data);

private:

E \_\_data;

DNode<E>\* \_\_Llink;

DNode<E>\* \_\_Rlink;

template <typename E> friend class LinkedBTree;

template <typename E> friend class LinkedQueue;

};

//DNode: 생성자와 메소드 정의

template <typename E>

DNode<E>::DNode(const E& data)

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

// class LinkedBTree

template <typename E>

class LinkedBTree {

public:

LinkedBTree();

DNode<E>\* makeLinkedBTree(const string str);

void Preorder(DNode<E>\* root) const;

void Inorder(DNode<E>\* root) const;

void Postorder(DNode<E>\* root) const;

void Levelorder(DNode<E>\* root)const;

private:

DNode<E>\* root;

};

//LinkedBTree class : 생성자와 소멸자

template <typename E>

LinkedBTree<E>::LinkedBTree()

: root(nullptr) {}

//이진 트리(수식 트리) 생성

template <typename E>

DNode<E>\* LinkedBTree<E>::makeLinkedBTree(const string str) {

stack<DNode<E>\*> s;

DNode<E>\* tNode;

for (int i = 0; i < str.length(); i++) {

//공백 제거

while (str[i] == ' ')

i++;

//새로운 노드를 생성

tNode = new DNode<E>(str[i]);

//연산자일 경우...

if (isOperator(str[i])) {

tNode->\_\_Rlink = s.top(); s.pop();

tNode->\_\_Llink = s.top(); s.pop();

}

s.push(tNode);

}

tNode = s.top();

s.pop();

//루트 노드 설정

root = tNode;

return tNode;

}

//깊이 우선 순회: 전위 순회

template <typename E>

void LinkedBTree<E>::Preorder(DNode<E>\* root)const {

if (root == nullptr)

return;

stack<DNode<E>\*> s;

s.push(root);

while (!s.empty()) {

DNode<E>\* tNode = s.top();

s.pop();

cout.width(3); cout << tNode->\_\_data;

if (tNode->\_\_Rlink) s.push(tNode->\_\_Rlink);

if (tNode->\_\_Llink) s.push(tNode->\_\_Llink);

}

}

//깊이 우선 순회: 중위 순회

template <typename E>

void LinkedBTree<E>::Inorder(DNode<E>\* root)const {

stack<DNode<E>\*> s;

DNode<E>\* tNode = root;

while (tNode || !s.empty()) {

while (tNode) {

s.push(tNode);

tNode = tNode->\_\_Llink;

}

tNode = s.top();

s.pop();

cout.width(3); cout << tNode->\_\_data;

tNode = tNode->\_\_Rlink;

}

}

//깊이 우선 순회: 후위 순회

template <typename E>

void LinkedBTree<E>::Postorder(DNode<E>\* root)const {

if (root == nullptr)

return;

stack<DNode<E>\*> s1, s2;

s1.push(root);

while (!s1.empty()) {

DNode<E>\* tNode1 = s1.top();

s1.pop();

s2.push(tNode1);

if (tNode1->\_\_Llink) s1.push(tNode1->\_\_Llink);

if (tNode1->\_\_Rlink) s1.push(tNode1->\_\_Rlink);

}

while (!s2.empty()) {

DNode<E>\* tNode2 = s2.top();

s2.pop();

cout.width(3); cout << tNode2->\_\_data;

}

}

//너비 우선 순회

template <typename E>

void LinkedBTree<E>::Levelorder(DNode<E>\* root)const {

if (root == nullptr)

return;

queue<DNode<E>\*> Q;

Q.push(root);

while (!Q.empty()) {

DNode<E>\* tNode = Q.front();

Q.pop();

cout.width(3);

cout << tNode->\_\_data;

if (tNode->\_\_Llink) Q.push(tNode->\_\_Llink);

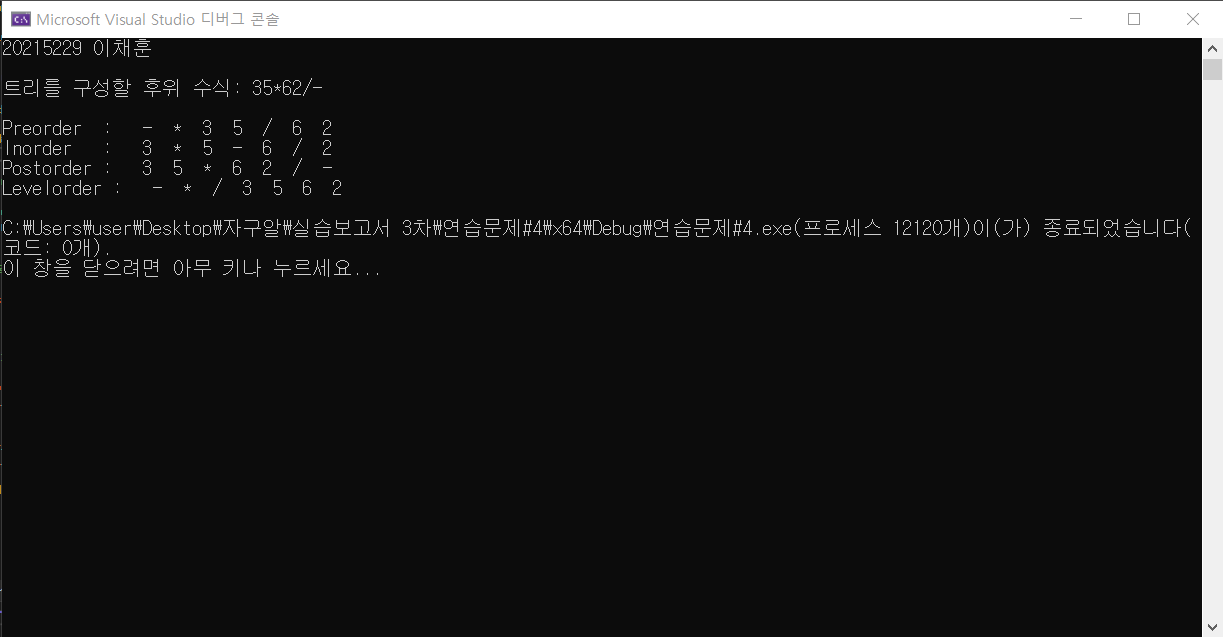
if (tNode->\_\_Rlink) Q.push(tNode->\_\_Rlink);

}

cout << endl;

}

**실행결과**



**연구조사**

**# is Operator()**

: 주어진 값이 연산자인지 판단하는 함수

**->** 주어진 값이 '+' 또는 '-' 또는 '\*' 또는 '/' 중 하나인지 확인하여, 맞으면 1을 반환하고, 아니면 0을 반환.

**# Preorder**

1. root를 먼저 방문한 후, 왼쪽 서브트리를 전체적으로 방문하고, 그 다음에 오른쪽 서브트리를 전체적으로 방문함.

2. 스택을 사용하여 노드를 처리하므로, 자식 노드의 방문 순서는 왼쪽에서 오른쪽임.

3. 출력 결과는 전위 순서대로 노드의 데이터가 나열됨.

- 조건 확인

: root 노드가 비어있는지 확인, 비어 있을 경우 함수를 종료함.

- 스택 초기화

: 노드의 순서를 기억하는데 사용됨

- 노드 처리

: 스택이 비어있지 않은 동안 계속 반복함. -> 현재 스택의 맨 위에 있는 노드를 가져옴->스택에서 노드 제거->노드의 데이터 출력

- 자식 노드 처리

: 오른쪽 자식이 있는 경우, 스택에 삽입/왼쪽 자식이 있는 경우, 스택에 삽입

**# Inorder**

1. 왼쪽 서브트리를 먼저 방문한 후 루프틑 방문하고, 그 다음에 오른쪽 서브트리를 방문함.

2. 스택을 사용하여 왼쪽 자식으로 이동하면서 부모 노드를 스택에 저장하고, 그 후에 부모 노드를 방문하여 출력함.

3. 출력 결과는 중위 순서대로 노드의 데이터가 나열됨.

- 루프 시작

: 현재 노드가 비어있지 않거나 스택이 비어있지 않은 동안 계속 반복함.

- 왼쪽 자식 방문

: 현재 노드가 비어있지 않은 동안 반복함. -> 현재 노드를 스택에 push -> 현재노드를 왼쪽 자식으로 이동

- 노드 처리

: 스택의 맨 위에 있는 노드를 가져옴. -> 스택에서 노드 제거

- 오른쪽 자식 방문

: 현재노드를 오른쪽 자식으로 이동

**# Postorder**

1. 왼쪽 서브트리를 먼저 방문한 후 오른쪽 서브트리를 방문하고, 그 다음에 루트를 방문함.

2. 두 개의 스택을 사용하여 루트를 기준으로 왼쪽 자식과 오른쪽 자식을 각각 다른 스택에 저장하고, 두번째 스택에서 노드를 꺼내어 출력함.

3. 출력 결과는 후위 순서대로 노드의 데이터가 나열됨.

- root노드 스택에 저장

: root 노드를 첫 번째 스택에 삽입

- 첫 번째 스택 이용하여 노드 저장

: 첫 번째 스택이 비어있지 않은 동안 계속 반복. -> 첫 번째 스택의 맨 위에 있는 노드를 가져옴. -> 첫 번째 스택에서 노드를 제거함. -> 두번째 스택에 노드를 넣음.

- 자식 노드 처리

: 왼쪽 자식이 있으면, 첫 번째 스택에 왼쪽 자식 push/오른쪽 자식이 있으면, 첫 번째 스택에 오른쪽 자식 push

- 두 번째 스택

: 두 번째 스택이 비어있지 않은 동안 계속 반복. -> 두 번째 스택의 맨 위에 있는 노드를 가져옴. -> 두 번째 스택에서 노드를 제거함.

**# Levelorder**

1. 같은 레벨의 노드를 왼쪽에서 오른쪽으로 순서대로 방문함.

2. 큐를 사용하여 레벨 단위로 노드를 저장하고 처리함.

3. 출력 결과는 레벨별로 노드의 데이터가 나열됨.

- root노드 큐에 저장

: root 노드를 큐에 삽입

- 큐를 이용하여 노드 저장

: 큐가 비어있지 않은 동안 계속 반복. -> 큐의 맨 앞에 있는 노드를 가져옴. -> 큐에서 노드를 제거함.

- 자식 노드 처리

: 왼쪽 자식이 있으면, 왼쪽 자식을 큐에 push/오른쪽 자식이 있으면, 오른쪽 자식을 큐에 push

**5. 힙 정렬: 알고리즘 구현**

**소스코드**

#include <iostream>

#include <random>

using namespace std;

#define arrMAXSIZE 15

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

template <typename E> void buildHeap(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) {

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;

}

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 << "20215229 이채훈\n" << endl;

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

buildHeap(arr, arrMAXSIZE);

cout << "buildheap: "; PRINT(arr, arrMAXSIZE);

HeapSort(arr, arrMAXSIZE);

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

return 0;

}

template <typename E>

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

int parent = root;

int child;

while ((parent \* 2) + 1 <= end) {

// 왼쪽 자식

child = (parent \* 2) + 1;

// 오른쪽 자식이 있고, 오른쪽 자식이 더 크다면 child를 오른쪽 자식으로 할당

if (child + 1 <= end && pArr[child] < pArr[child + 1]) {

++child;

}

// 부모가 자식보다 작으면 교환

if (pArr[parent] < pArr[child]) {

SWAP(&pArr[parent], &pArr[child]);

parent = child;

}

else {

break;

}

}

}

template <typename E>

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

// 마지막 비닐 노드의 인덱스 계산

int startIdx = (num / 2) - 1;

// 마지막 비닐 노드부터 루트까지 힙 속성을 유지하면서 반복

for (int i = startIdx; i >= 0; --i) {

percolateDown(pArr, i, num - 1);

}

}

template <typename E>

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

// 1. 힙 생성

buildHeap(pArr, num);

// 2. 힙 정렬

for (int i = num - 1; i > 0; --i) {

// 최대 값을 배열 끝으로 이동

SWAP(&pArr[0], &pArr[i]);

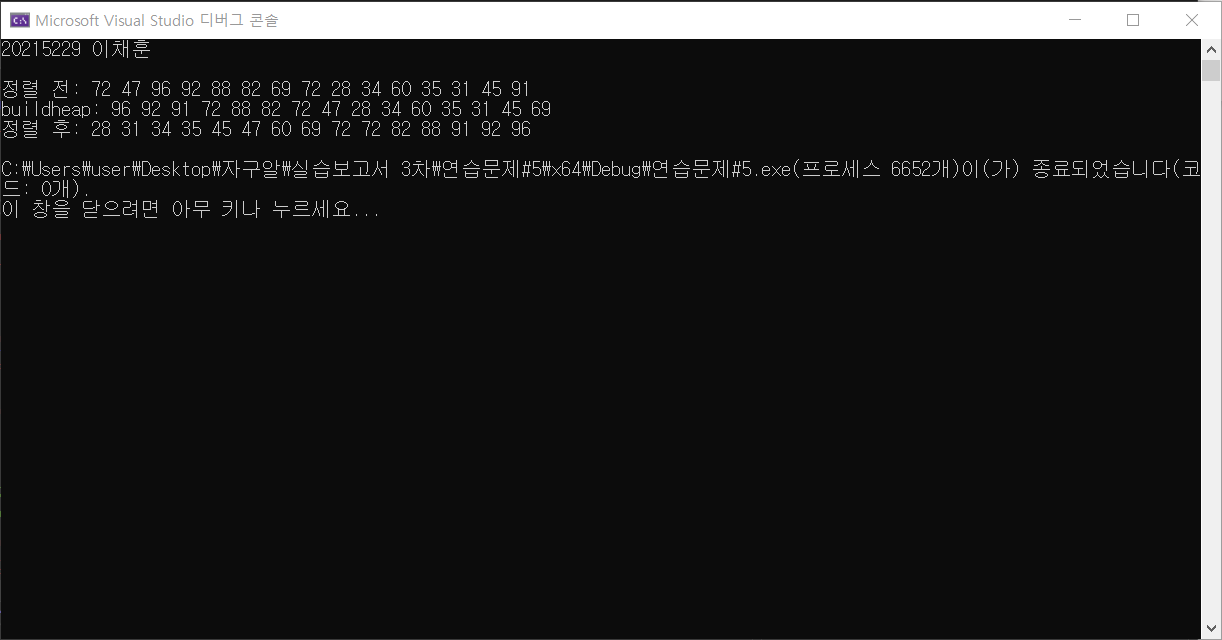
// 힙 속성 유지하면서 힙 크기 감소

percolateDown(pArr, 0, i - 1);

}

}

**실행결과**

****

**연구조사**

**# Heap Sort 동작**

1. 초기 배열을 최대 힙으로 만듦(buildHeap).

2. 힙에서 최대값(루트)을 꺼내어 배열의 마지막 요소와 교환함.

3. 힙 크기를 하나 감소시키고, 교환된 값이 힙 속성을 위배하지 않도록 percolateDown을 호출하여 힙을 재조정함.

-> 위 과정을 반복하여 배열 정렬

**# Heap Sort 특징**

1. 시간 복잡도 : O(nlogn)

2. 안정적인 정렬 알고리즘 X

3. 힙 정렬은 추가적인 메모리가 필요하지 않고, 특히 입력 데이터의 크기가 매우 큰 경우에 유용함.

4. 최대 힙의 경우, 부모 노드는 항상 자식노드보다 크거나 같음.(실습 : 최대 힙)

5. 최소 힙의 경우, 부모 노드는 항상 자식노드보다 작거나 같음.

**# percolateDown**

: 힙 속성을 유지하면서 주어진 노드에서부터 아래로 내려가며 힙을 재조정함. 주어진 노드의 자식 중 큰 값을 가진 자신과 교환하면서 아래로 내려가면서 힙 속성을 유지함.

**- Paramter**

pArr : 정렬 대상 배열

root : 현재 부모 노드의 인덱스

end : 힙의 끝 인덱스

**- 자식 노드 선택**

: 부모노드의 왼쪽 자식이 힙에 속할 동안 반복 -> 현재 부모 노드의 왼쪽 자식을 선택함.

**- 오른쪽 자식과 비교**

: 오른쪽 자식이 있고, 오른쪽 자식이 왼쪽 자식보다 크면, child를 오른쪽 자식으로 할당

**- 부모와 자식 비교**

: 부모가 자식보다 작으면 교환함. -> 부모가 자식보다 크거나 같으면 종료.

**- 힙 속성 유지**

: 교환된 경우, 현재 자식 위치로 이동하고, 다음 레벨에서 반복하여 힙 속성 유지.

**# buildHeap**

**- Parameter**

pArr : 정렬 대상 배열

num : 배열의 크기

- **함수동작**

배열을 마지막 비닐 노드에서부터 루트 노드까지 각 노드에 대해 percolateDown을 호출하여 힙 속성을 유지하면서 거슬러 올라감. -> 마지막 비닐 노드부터 시작하여 힙 속성을 만족하도록 노드들을 조정함. -> 최종적으로, 배열 전체가 최대 힙이 되도록 함.