9.1 우선순위 큐 추산 데이터 타입

우선순위 큐의 소개 우선순위의 개념을 큐에 도입

데이터들이 우선순위를 가지고 있고, 우선순위가 높은 데이터가 먼저 나감

히프로 구현

우선순위 큐 추상자료형 객체: n개의 element형의 우선순위를 가진 요소들의 모임

연산:

create() ::= 우선순위 큐를 생성한다.

init(q) ::= 우선순위 큐 q를 초기화한다,

is\_empty(q) ::= 우선순위 큐가 비어있는지를 검사한다

is\_full(q) ::= 우선순위 큐가 가득 찼는가를 검사한다

insert(q,x) ::= 우선순위 큐에 요소 x를 추가한다

delete(q) ::= 우선순위 큐로부터 가장 우선순위가 높은 요소를 삭제하고 이 요소를 반환한다

find(q) ::= 우선순위가 가장 높은 요소를 반환한다

우선순위 큐의 종류 최소 우선순위 큐 가장 우선순위가 낮은 요소를 먼저 삭제

최대 우선순위 큐 가장 우선순위가 높은 요소를 먼저 삭제

Quiz 1. 스택, 큐를 우선순위 큐를 이용해 구현할 수 있기 때문

2. 있음

9.2 우선순위 큐의 구현 방법

배열을 사용하는 방법 삽입 배열의 맨 끝에 새로운 요소를 추가

O(1)

삭제 우선순위가 가장 높은 요소를 찾아 삭제

O(n)

연결리스트을 사용하는 방법 삽입 연결리스트의 첫 노드로 삽입

O(n)

삭제 우선순위가 가장 높은 요소를 찾아 삭제

O(n)

히프을 사용하는 방법 히프 완전이진트리의 일종

일종의 느슨한 정렬상태 유지 정렬이 된 것도, 안된 것도 아닌 상태

삽입 O(logn)

삭제 O(logn)

Quiz 1. for (i = 0-- > 10)

it = 제일후방

삽입(요소, it)

for(i=0-->10)

max=최고우선순위

삭제(max)

2. 삽입 O(1)

삭제 O(n)

9.3 히프

히프의 개념 완전이진트리 기반

히프 여러 개의 값들 중에서 가장 큰 값이나 가장 작은 값을 빠르게 찾아내도록 만들어진 자료구조

부모노드의 키값이 자식노드의 키값보다 항상 큰 이진 트리 = 느슨한 정렬상태

히프의 종류 최대히프 부모노드의 키값이 자식노드의 키값보다 크거나 같음

최소히프 부모노드의 키값이 자식노드의 키값보다 작거나 같음

히프의 구현 히프 = 완전 이진트리 🡪 배열로 구현 가능

이진트리의 노드에 번호를 붙여, 그 번호를 인덱스로 함

배열의 첫번째 인덱스는 쓰지 않음

9.4 히프의 구현

히프의 정의 1차원 배열로 표현 가능 정의 #define MAX\_ELEMENT 200

typedef struct {

int key;

} element;

typedef struct {

element heap[MAX\_ELEMENT];

int heap\_size;

} HeapType;

삽입연산 새로운 노드를 마지막 노드로 삽입함 🡪 새로운 노드를 부모노드와 교환

알고리즘9.1 insert\_max\_heap(A, key) {

heap\_size = heap\_size + 1;

i = heap\_size;

A[i] = key;

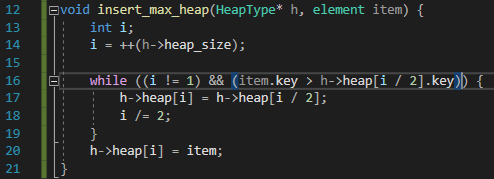
while (i != 1 and A[i] > A[PARENT(i)]) {

swap(A[i], A[PARENT]);

i = PARENT(i);

}

}

프로그램9.1 

히프의 삭제연산 최대값을 가진 요소 삭제 = 루트노드 삭제

삭제 후, 히프의 재구성 루트노드 삭제 🡪 히프의 마지막 노드를 가져옴 🡪 루트노드의 자식노드와 교환 🡪 교환된 노드(전 루트노드)와 자식노드 비교 🡪 교환

알고리즘9.2 delete\_max\_heap(A) :

item = A[i];

heap\_size = A[heap\_size];

i = 2;

while (i <= heap\_size) {

if (i<heap\_size and A[i + 1] > A[i])

then largest = i + 1;

else largest = i;

if (A[PARENT(largest)] > A[largest])

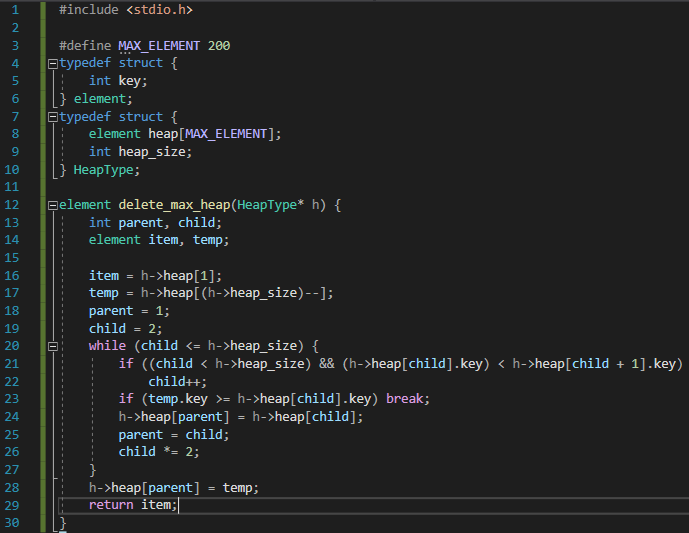
then break;

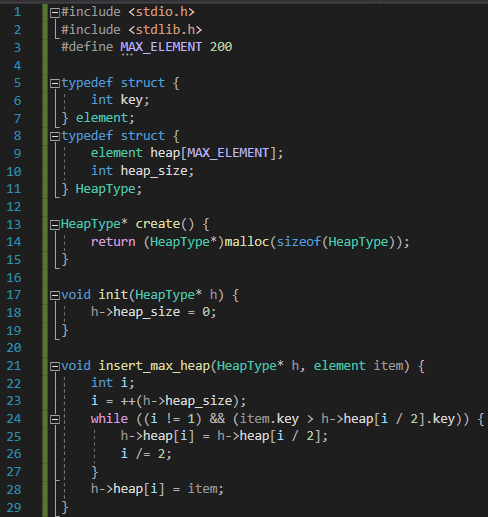
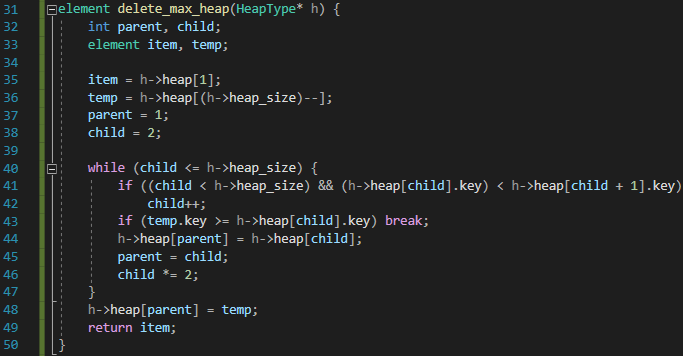
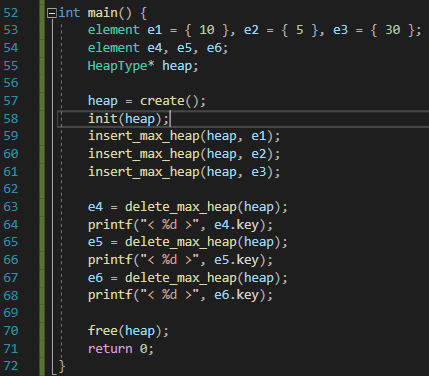
swap(A[PARENT(largest)], A[largest]);

i = CHILD(largest);

}

return item;

프로그램9.2 

전체 프로그램 프로그램9-3   

히프의 복잡도 분석 삽입의 시간 복잡도 O(log­2n)

삭제의 시간 복잡도 O(log­2n)

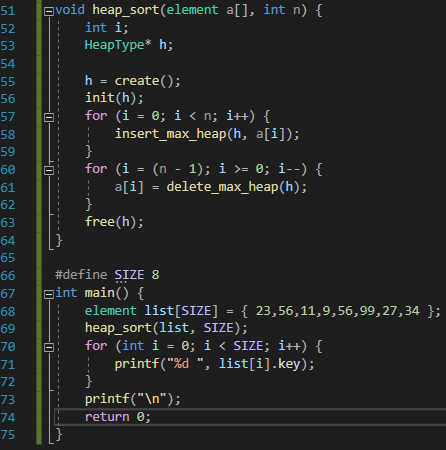
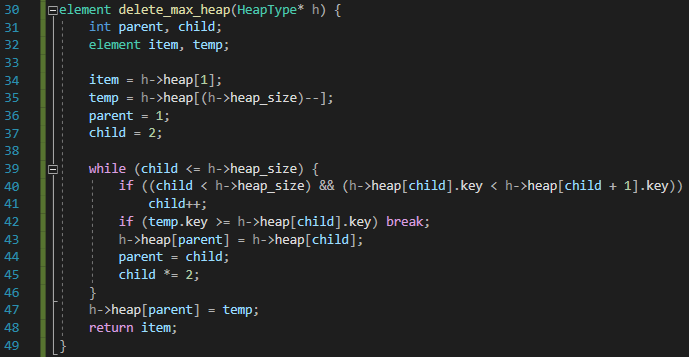
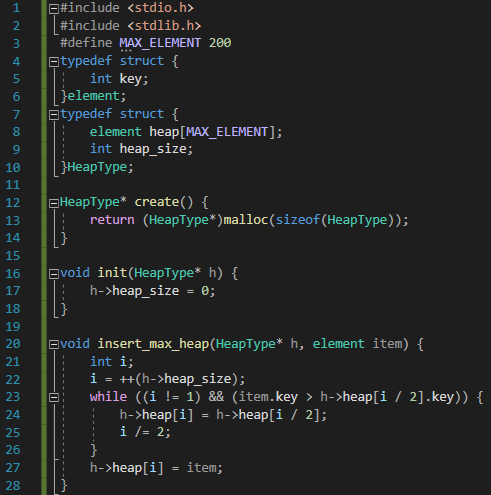
Quiz 1. 12 111 8 10 6 2 5 3 4

2. 10 6 8 3 4 2 5

9.5 히프 정렬

최대 히프를 이용한 정렬 n개의 요소가 O(log­­2n)시간 안에 정렬

최대 히프 생성 🡪 루트노드부터 거꾸로 배열의 뒤쪽부터 저장하면 됨

히프의 정렬의 구현 프로그램9.4 

히프의 정렬의 복잡도 시간복잡도 O(log­­2n)

히프 정렬이 유용한 경우 가장 큰 값 몇 개가 필요할 때

Quiz 1. #include <stdio.h>

#include <stdlib.h>

#define MAX\_ELEMENT 200

typedef struct {

int key;

}element;

typedef struct {

element heap[MAX\_ELEMENT];

int heap\_size;

}HeapType;

HeapType\* create() {

return (HeapType\*)malloc(sizeof(HeapType));

}

void init(HeapType\* h) {

h->heap\_size = 0;

}

void insert\_min\_heap(HeapType\* h, element item) {

int i;

i = ++(h->heap\_size);

while ((i != 1) && (item.key < h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item;

}

element delete\_min\_heap(HeapType\* h) {

int parent, child;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

parent = 1;

child = 2;

while (child <= h->heap\_size) {

if ((child < h->heap\_size) && (h->heap[child].key > h->heap[child + 1].key))

child++;

if (temp.key <= h->heap[child].key) break;

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

return item;

}

void heap\_sort(element a[], int n) {

int i;

HeapType\* h;

h = create();

init(h);

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

insert\_min\_heap(h, a[i]);

}

for (i = (n - 1); i >= 0; i--) {

a[i] = delete\_min\_heap(h);

}

free(h);

}

#define SIZE 8

int main() {

element list[SIZE] = { 23,56,11,9,56,99,27,34 };

heap\_sort(list, SIZE);

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

printf("%d ", list[i].key);

}

printf("\n");

return 0;

}

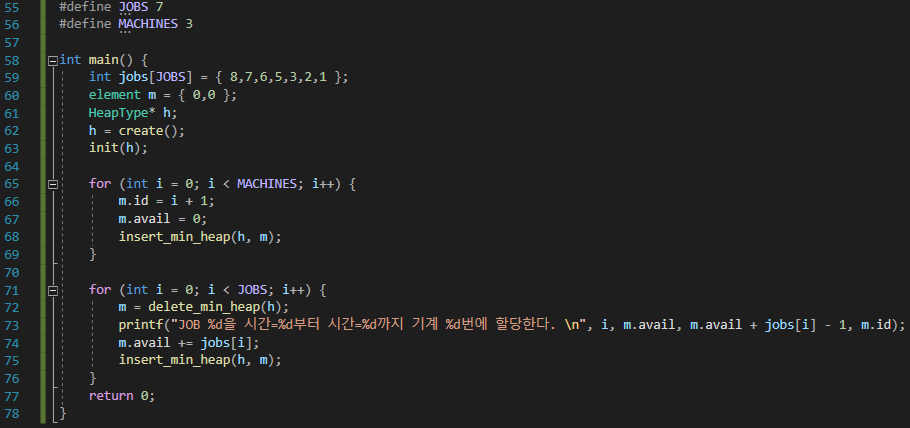
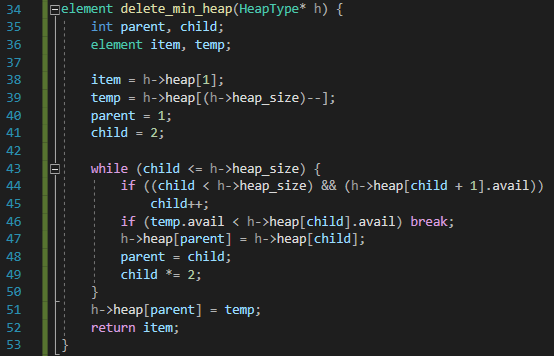
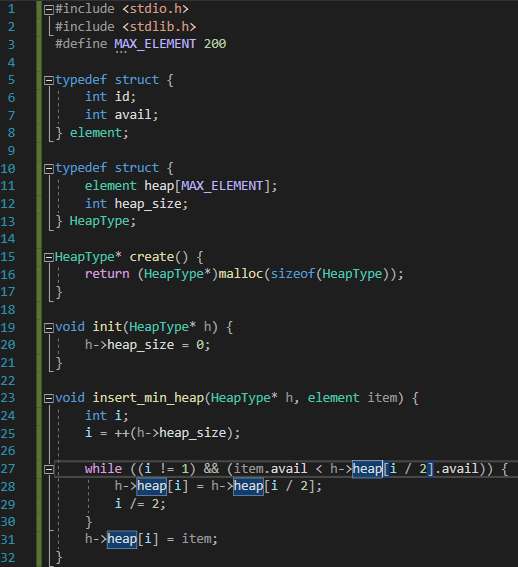
9.6 머쉰 스케줄링

머신 스케줄링 모든 기계를 가동해 가장 최소의 시간에 작업들을 모두 끝내는 것

LPT 가장 긴 작업을 우선적으로 기계에 할당

최소히프 사용 히프에 모든 기계의 종료 시간 저장

히프에서 최소의 종료시간을 가지는 기계 삭제 후 작업 할당

LPT알고리즘의 구현 프로그램9.5 

9.7 허프만 코드

허프만 코딩 트리 메시지의 내용을 압축하는데 사용됨

빈도수 영문 텍스트에서 해당 글자가 나타나는 횟수

가변길이 비트열 사용 빈도수가 낮은 문자에 더 긴 비트열 사용 🡪 메모리 절약

각각의 글자를 어떤 비트로 표현했는지 알려주는 테이블이 있어야 함

허프만코드 가변길이 비트열을 만들기 위한 이진트리

생성 1. 비트수를 계산해 리스트를 만듬 코드길이\*빈도수=비트수

ex) 4, 6, 8, 12, 15

2. 가장 작은 비트수 2개를 자식노드로 트리를 만듬, 루트노드는 자식노드의 합, 두 자식노드를 리스트에서 없애고 루트노드를 리스트에 집어넣고 정렬 ex) 8, 10, 12, 15

10

4 6

3. 그 다음 작은 비트수 1개를 자식노드로 함, 루트노드는 자식노드의 합, 두 자식노드를 리스트에서 없애고 루트노드를 리스트에 집어넣고 정렬 ex) 12, 15, 18

18

10

8 4 6

4. 그 다음 작은 비트수 2개를 자식노드로 함, 루트노드는 자식노드의 합, 두 자식노드를 리스트에서 없애고 루트노드를 리스트에 집어넣고 정렬 ex) 18, 27

18

10 27

8 4 6 12 15

5. 남은 두 값을 자식노드로 함, 루트노드는 자식노드의 합, 두 자식노드를 리스트에서 없앰 ex) 45

18

10 27

8 4 6 12 15

6. 허프만 트리 읽기 왼쪽 간선 비트 1

오른쪽간선 비트 0

ex) 45

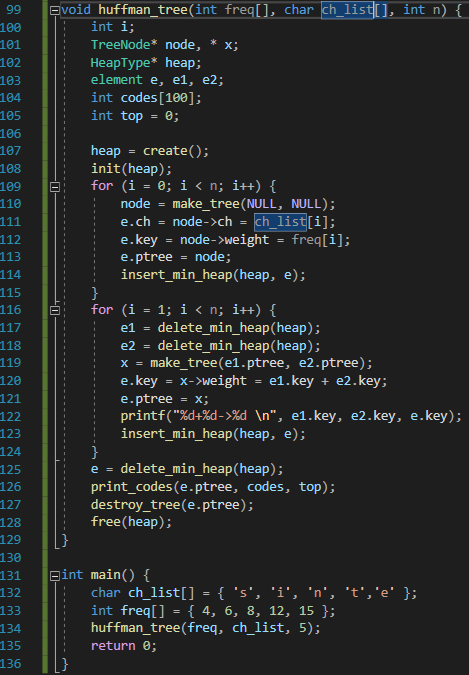
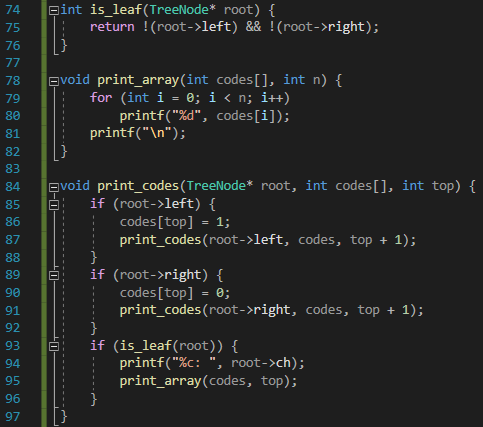
18

10 27

8 4 6 12 15

빈도수 6 = 100

허프만트리에서 가장 작은 빈도수를 얻는 과정 최소 히프 이용

프로그램9.6 

연습문제

1. 1

2. 3

3. 2

4. 1

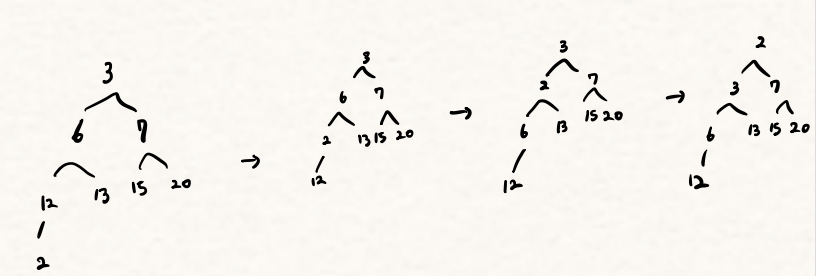
5. 2

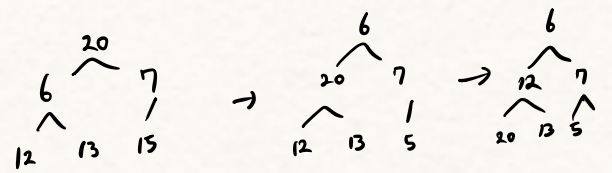
6. 2번, 3번 노드 중 하나

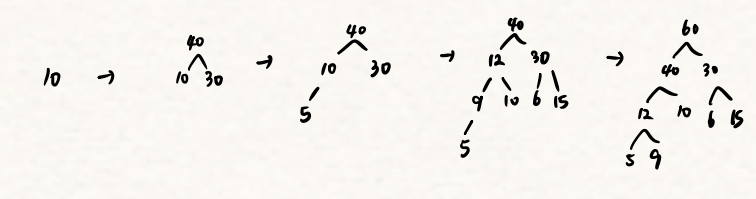
7. 4

8. (1) 3 6 7 9 15 18 14

(2) 2 6 3 7 15 18 7 14 9

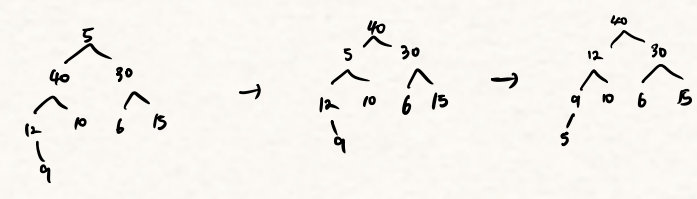
9. (1) 

(2) 

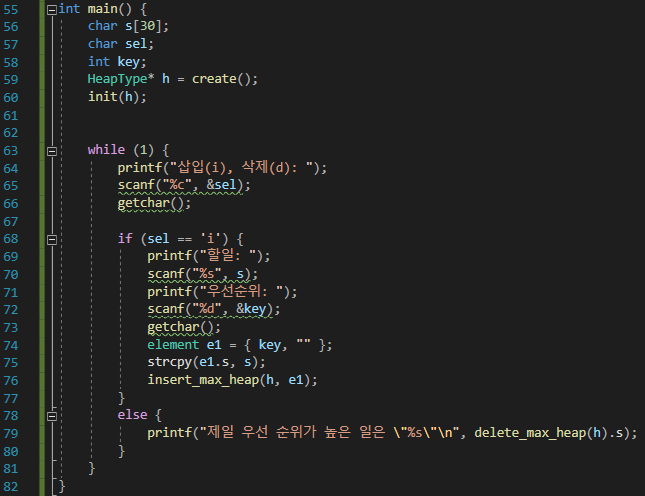
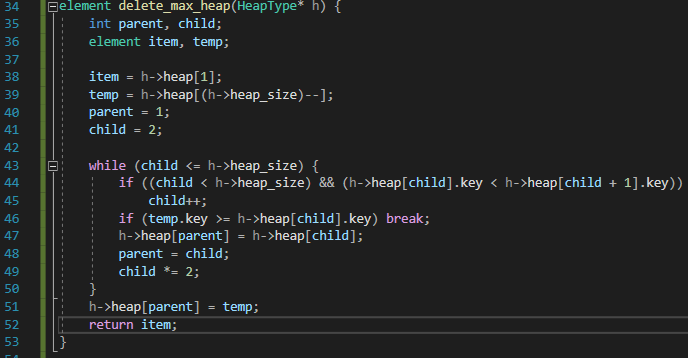
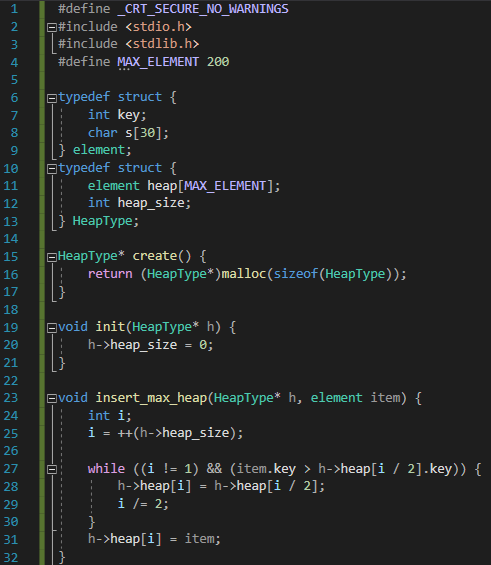
10. (1) 

(2) 0 1 2 3 4 5 6 7 8 9

60 40 30 12 10 6 15 5 9

(3) 

11.



12. 최소 히프트리가 아님 6의 자식노드가 5임

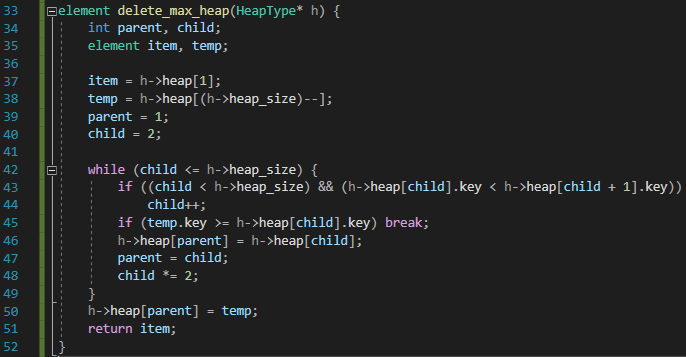
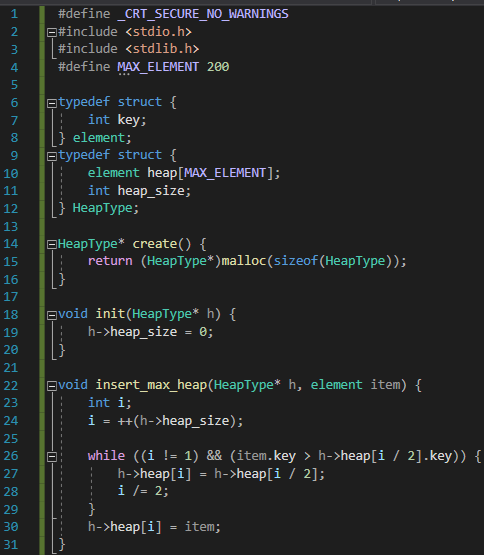
13.

1

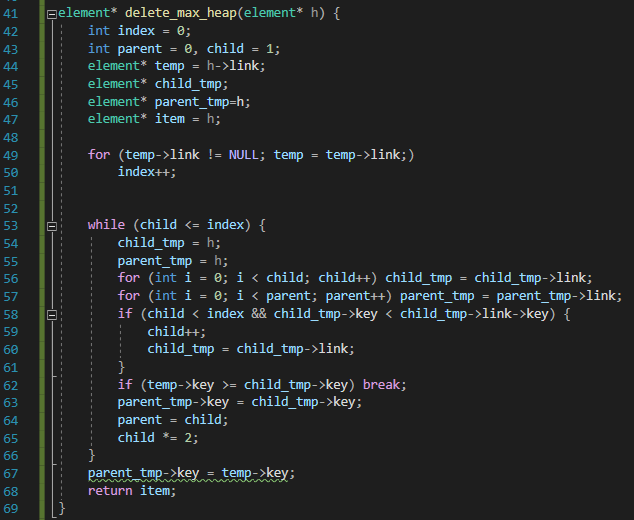
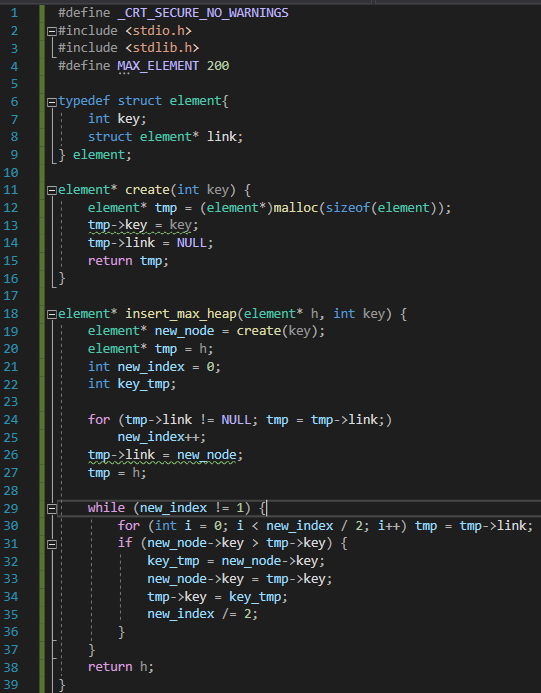
5 12

20 15

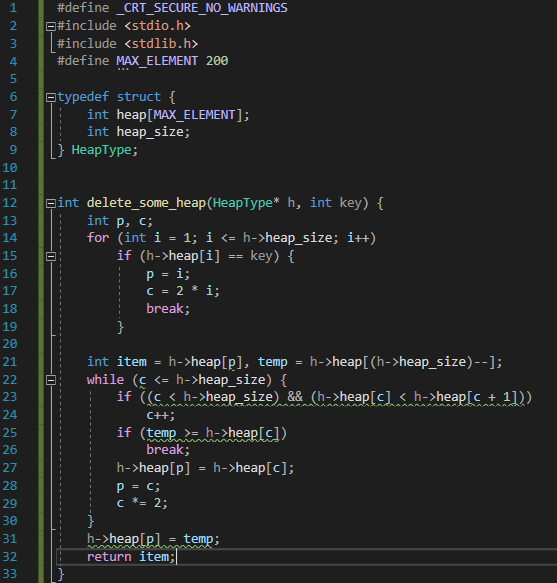
14.



15.



16.



17.

