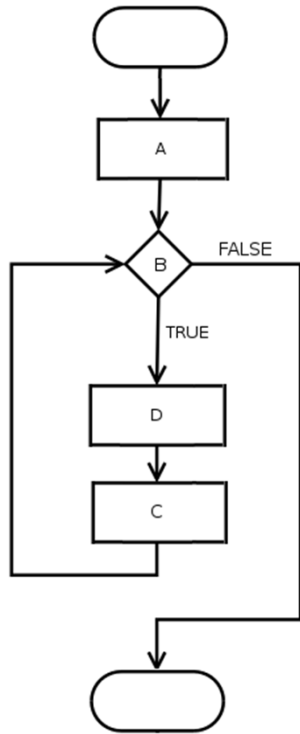


자료구조 & 알고리즘

for(A;B;C)
D;

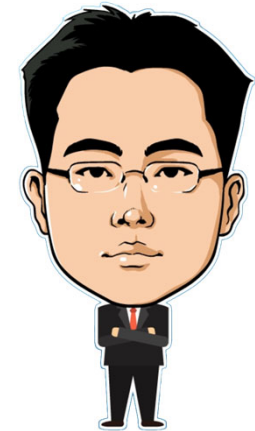


자료구조
(Data Structures)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



백문이불여일타(百聞而不如一打)



목 차

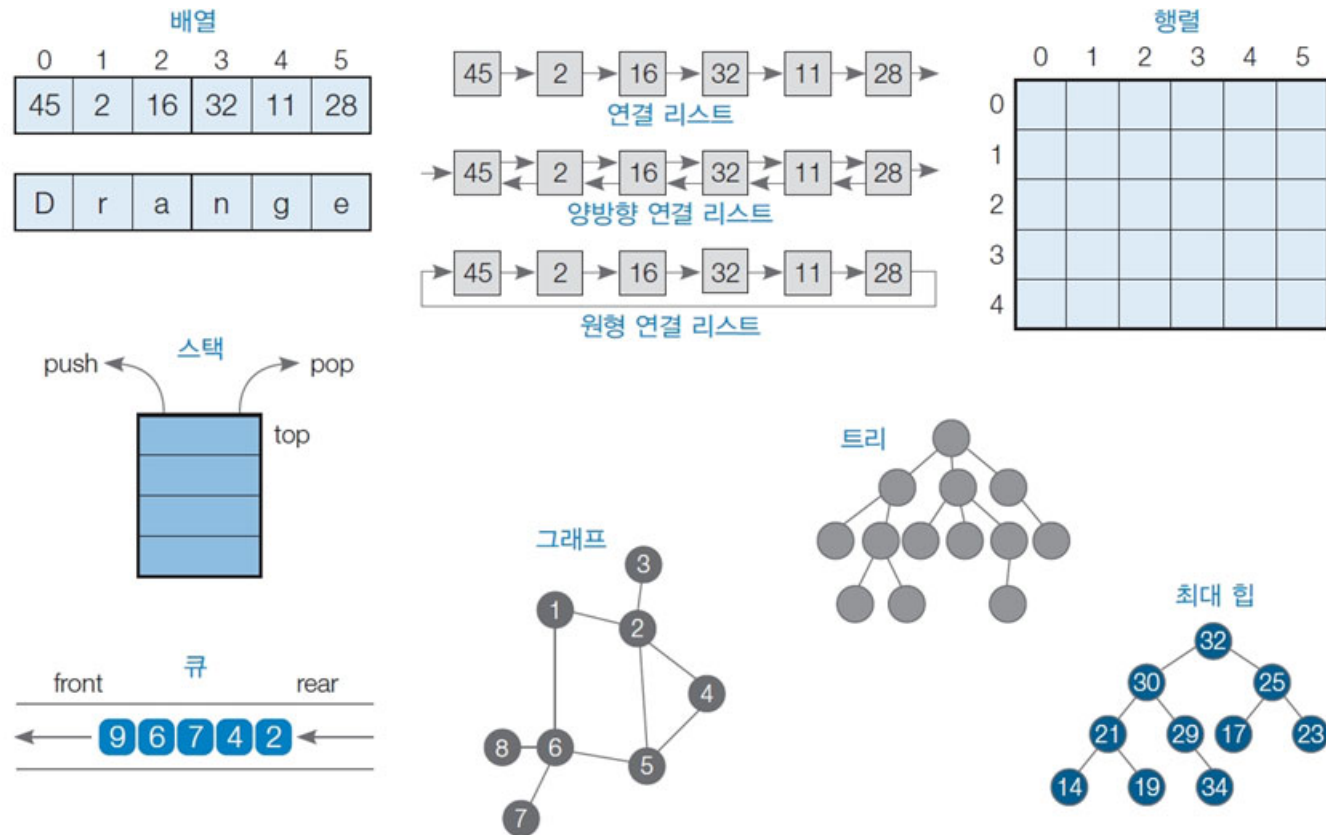


- 배열과 리스트
- 스택과 큐
- 트리와 검색 트리
- 해시 테이블



자료구조

● 자료구조의 종류

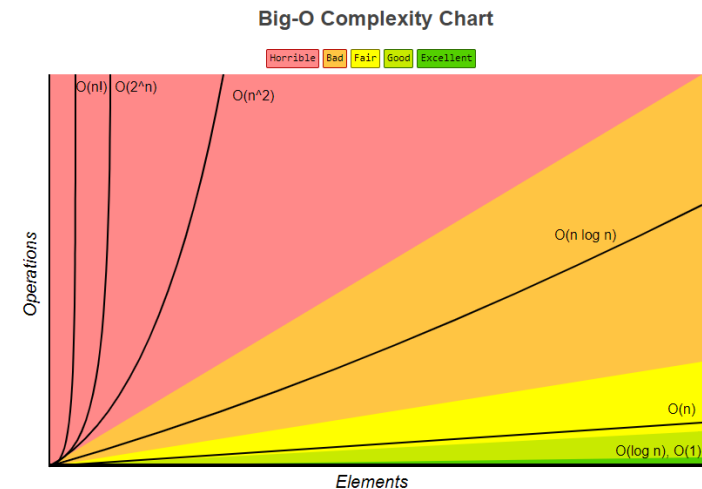


[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

자료구조: 시간 복잡도

● 자료 구조: 시간 복잡도

- 배열 또는 연결 리스트: $O(N)$, 평균 $\Theta(N)$
- 이진 검색 트리: 검색, 삽입, 삭제 시 평균 $\Theta(\log N)$, 최악의 경우 $O(N)$
- 균형 이진 검색 트리
 - 검색, 삽입, 삭제 시 최악의 경우 $O(\log N)$
 - AVL 트리, RB 트리
- 균형 다진 검색 트리
 - 검색, 삽입, 삭제 시 최악의 경우 $O(\log N)$
 - 2-3 트리, 2-3-4 트리, B-트리
- 해시 테이블
 - 검색, 삽입, 삭제 시 평균 $O(1)$



배열과 리스트



- 배열과 리스트

백문이불여일타(百聞而不如一打)

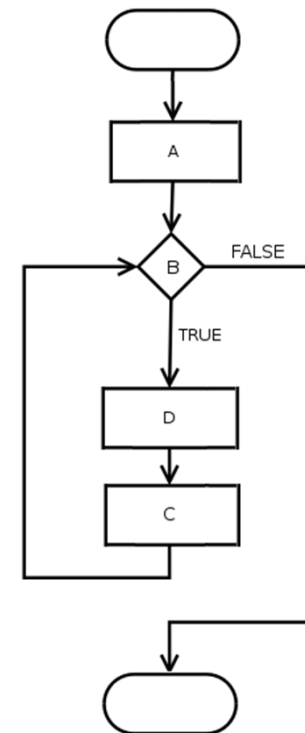
- 정적 배열과 동적 배열

- 스택과 큐

- 트리와 검색 트리

- 해시 테이블

for(A;B;C)
D;



정적 배열과 동적 배열 (1/2)

예제 2-1: 파이썬 내장 함수와 리스트 조작 함수

| Python

```
# 리스트 객체 생성
sList = [10, 20, 30, 40, 50]

# 내장 함수: len
print(f'전체 원소: {sList}')
```

```
print(f'전체 원소 개수: {len(sList)}')
```

```
# 리스트 조작 함수 (append): 리스트 맨 마지막 원소로 추가
sList.append(60)
sList.append(70)

# 내장 함수: len
print(f'전체 원소: {sList}')
```

```
print(f'전체 원소 개수: {len(sList)}')
```

```
# 리스트 조작 함수 (count): 특정 원소의 총 개수
print(f'데이터 10의 총 개수: {sList.count(10)}')
```

```
print(f'데이터 20의 총 개수: {sList.count(20)}')
```

```
print(f'데이터 30의 총 개수: {sList.count(30)}')
```

```
# 내장 함수: len, sum
print(f'전체 원소 합계: {sum(sList)}')
```

```
print(f'전체 원소 평균: {sum(sList)/len(sList)}')
```

```
# 내장 함수: max, min
print(f'전체 원소 최대값: {max(sList)}')
```

```
print(f'전체 원소 최솟값: {min(sList)}')
```

```
IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
Python 3.11.2 (tags/v3.11.2:878ead1, Feb
Type "help", "copyright", "credits" or "li
>>>
===== RESTART: C:\Users\WclickW
전체 원소: [10, 20, 30, 40, 50]
전체 원소 개수: 5
전체 원소: [10, 20, 30, 40, 50, 60, 70]
전체 원소 개수: 7
데이터 10의 총 개수: 1
데이터 20의 총 개수: 1
데이터 30의 총 개수: 1
전체 원소 합계: 280
전체 원소 평균: 40.0
전체 원소 최대값: 70
전체 원소 최솟값: 10
>>>
```

정적 배열과 동적 배열 (2/2)

예제 2-2: 정적 배열과 C++ STL 알고리즘 -- sort

| C++

```
#include <iostream>
#include <iomanip>      // setw
#include <algorithm>    // sort
using namespace std;
// using std::sort;

int main(void)
{
    // 정적 배열
    int arr[] = { 45, 23, 36, 87, 56 };
    int arrSize = sizeof(arr) / sizeof(*arr);

    cout << "정렬 전: ";
    for (int *p = arr; p < arr + arrSize; p++)
        cout << setw(3) << *p;
    cout << endl;

    // C++ STL 알고리즘: sort
    sort(arr, arr + arrSize);

    cout << "정렬 후: ";
    for (int num : arr)
        cout << setw(3) << num;
    cout << endl;
    return 0;
}
```

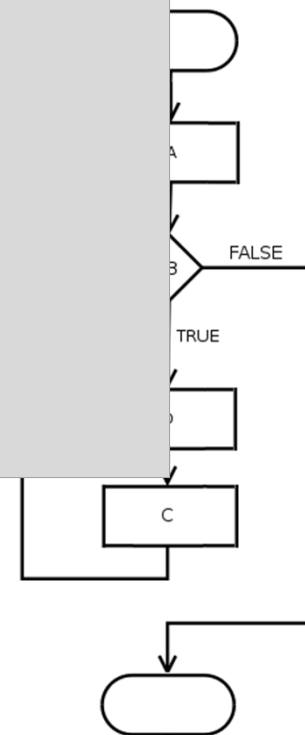
Microsoft Visual Studio 디버그 × + ▾

정렬 전: 45 23 36 87 56
정렬 후: 23 36 45 56 87

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...

정적 배열과 동적 배열

파이썬 내장 클래스: list



리스트 조작 함수 (1/4)

- 리스트 조작 함수: 리스트 항목 추가 또는 삭제

- 리스트 항목 추가 또는 삭제

함 수	세부 내용	
index	list.index(elmnt)	리스트에서 지정된 항목의 위치를 반환한다.
count	list.count(value)	리스트에서 지정된 값의 개수를 반환한다.
append	list.append(elmnt)	리스트 맨 뒤에 새로운 항목을 추가한다.
insert	list.insert(pos, elmnt)	리스트의 지정된 위치에 새로운 항목을 추가한다.
extend	list.extend(iterable)	리스트의 맨 마지막에 새로운 리스트를 추가(연결)한다. 즉, 리스트의 '+' 연산과 동일하다.
remove	list.remove(elmnt)	리스트에서 지정된 항목을 삭제한다. 단, 첫 번째로 검색된 항목(값)만 지운다.
pop	list.pop(pos)	리스트에서 지정된 항목을 삭제한다(기본값은 -1).
clear	list.clear()	리스트의 모든 내용을 삭제한다.

리스트 조작 함수 (2/4)

- 리스트 조작 함수: 리스트 복사

- **copy** 함수: 리스트의 모든 항목(값)을 새로운 리스트에 복사한다.

함 수	세부 내용	
copy	<code>list.copy()</code>	리스트의 모든 항목을 새로운 리스트에 복사 한다.

```
# 리스트 객체 생성 후 모든 원소를 새로운 리스트에 복사
```

```
sList = [ '국어', '영어', '수학', '과학' ]
```

```
temp = sList.copy() # 새로운 리스트 객체 temp 생성
```

```
sList = [ 1, 2, 3, 4, 5 ]
```

```
# sList의 복사본인 새로운 리스트(aList, bList, cList) 객체 생성
```

```
aList = sList[:]
```

```
bList = sList.copy()
```

```
cList = list(sList)
```

리스트 조작 함수 (3/4)

- 리스트 조작 함수: 리스트 복사

- deepcopy 함수: 깊은 복사(deep copy)를 수행한다.

- 얕은 복사(shallow copy)

```
sList = [ 1, 2, 3, 4, 5, [6, 7, 8, 9, 0] ]
```

```
# sList의 복사본인 새로운 리스트 aList, bList, cList 객체 생성(얕은 복사)
```

```
aList = sList[:]
```

```
bList = sList.copy()
```

```
cList = list(sList)
```

- 깊은 복사(deep copy)

```
# sList의 복사본인 새로운 리스트 dList 객체 생성(깊은 복사)
```

```
import copy
```

```
dList = copy.deepcopy(sList)
```

리스트 조작 함수 (4/4)

● 리스트 조작 함수: 정렬과 역순

○ 리스트 정렬과 역순

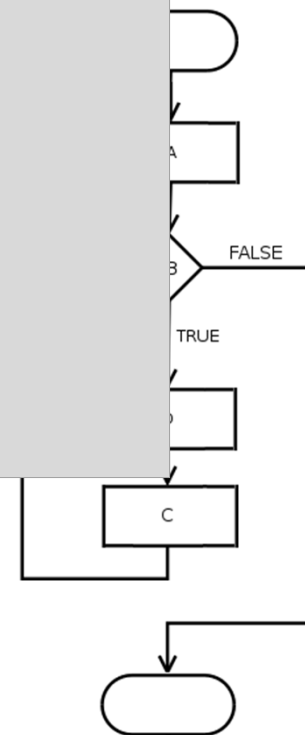
함 수	세부 내용	
sort	list.sort()	리스트 항목을 내부적으로 정렬한다(기존 순서 변경).
	list.sort(reverse=True False, key=myFunc) reverse=False (기본값) / reverse=True (내림차순) key : 정렬 기준을 지정한다.	
reverse	list.reverse()	리스트 항목의 순서를 반대로 바꾼다(기존 순서 변경).

○ sorted 내장 함수

- 리스트의 항목을 정렬하여, 새로운 리스트를 반환한다(복사본 생성).
 - 단, 기존 리스트 항목들의 순서는 변경되지 않는다.

정적 배열과 동적 배열

C++ STL: vector 클래스



vector 클래스 (1/6)

예제 2-3: vector 클래스 조작 함수 -- 동적 배열

(1/4)

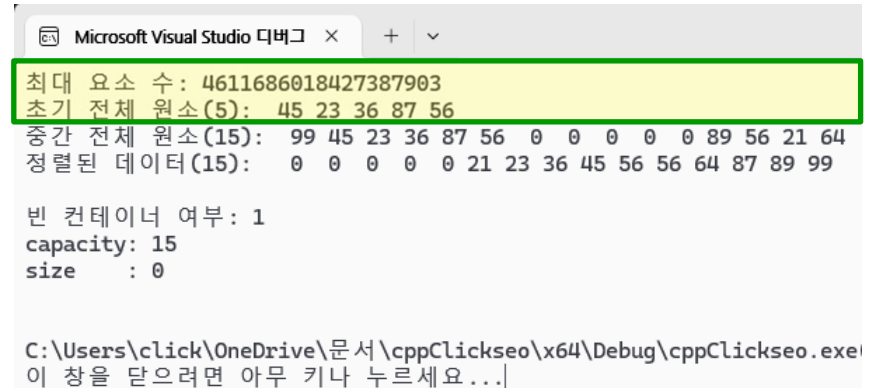
```
#include <iostream>
#include <iomanip>      // setw
#include <vector>       // vector
#include <algorithm>    // sort
using namespace std;
// using std::sort;
```

```
int main(void)
{
```

```
    // int          arr[] = { 45, 23, 36, 87, 56 };      // 정적 배열
    vector<int>     v = { 45, 23, 36, 87, 56 };          // 동적 배열
```

```
    // max_size: 시스템 또는 라이브러리 구현 제한으로 인해
    //           컨테이너가 보유할 수 있는 최대 요소 수를 반환
    cout << "최대 요소 수: " << v.max_size() << endl;
```

```
    // capacity: 현재 공간을 할당한 요소 수를 반환
    cout << "초기 전체 원소 (" << v.capacity() << ") : ";
    for (int num : v)
        cout << setw(3) << num;
    cout << endl;
```



Microsoft Visual Studio 디버그

최대 요소 수: 4611686018427387903
초기 전체 원소(5): 45 23 36 87 56
중간 전체 원소(15): 99 45 23 36 87 56 0 0 0 0 0 89 56 21 64
정렬된 데이터(15): 0 0 0 0 0 21 23 36 45 56 56 64 87 89 99

빈 컨테이너 여부: 1
capacity: 15
size : 0

C:\Users\click\OneDrive\문서\cppClickseo\x64\Debug\cppClickseo.exe
이 창을 닫으려면 아무 키나 누르세요...

vector 클래스 (2/6)

예제 2-4: vector 클래스 조작 함수 -- 동적 배열

(2/4)

```
// resize: 컨테이너의 크기를 조정한다.  
v.resize(10);  
v.push_back(89);  
v.push_back(56);  
v.push_back(21);  
v.push_back(64);  
// push_back: 주어진 값을 맨 끝에 추가한다.  
// v.pop_back();  
// pop_back: 마지막 요소를 제거한다.
```

```
// insert: 지정된 위치에 요소를 삽입한다.
```

```
auto vi = v.begin();  
vi = v.insert(vi, 99);  
// v.erase(v.end());
```

```
// erase: 지정된 요소를 지운다.
```

```
// at: 지정된 위치의 요소에 대한 참조를 반환한다.  
// front: 첫 번째 요소에 대한 참조를 반환한다.  
// back: 마지막 요소에 대한 참조를 반환한다.
```

```
cout << "중간 전체 원소 (" << v.capacity() << ") : ";  
for (int i = 0; i < v.size(); i++) // size: 요소 수를 반환  
    cout << setw(3) << v.at(i);  
cout << endl;
```

```
Microsoft Visual Studio 디버그 x + v  
최대 요소 수: 4611686018427387903  
초기 전체 원소(5): 45 23 36 87 56  
중간 전체 원소(15): 99 45 23 36 87 56 0 0 0 0 0 89 56 21 64  
정렬된 데이터(15): 0 0 0 0 0 21 23 36 45 56 56 64 87 89 99  
  
빈 컨테이너 여부: 1  
capacity: 15  
size : 0  
  
C:\Users\click\OneDrive\문서\cppClickseo\x64\Debug\cppClickseo.exe  
이 창을 닫으려면 아무 키나 누르세요...
```

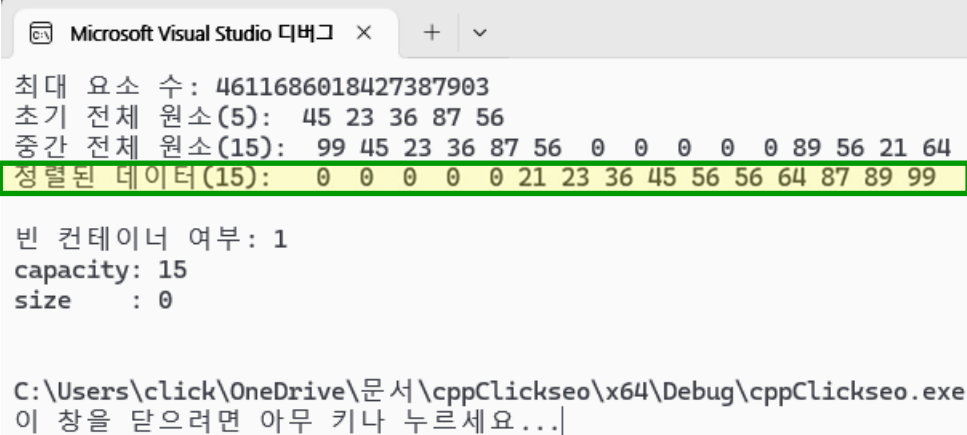
vector 클래스 (3/6)

예제 2-5: vector 클래스 조작 함수 -- 동적 배열

(3/4)

```
// C++ STL 알고리즘: sort
sort(v.begin(), v.end());

// 반복자(iterator): 컨테이너에 저장된 원소를 순회하고 접근
vector<int>::iterator p;
cout << "정렬된 데이터 (" << v.capacity() << ") : ";
for (p = v.begin(); p != v.end(); p++)
    cout << setw(3) << *p;
cout << endl;
```



Microsoft Visual Studio 디버그

최대 요소 수: 4611686018427387903
초기 전체 원소(5): 45 23 36 87 56
중간 전체 원소(15): 99 45 23 36 87 56 0 0 0 0 0 89 56 21 64
정렬된 데이터(15): 0 0 0 0 0 21 23 36 45 56 56 64 87 89 99

빈 컨테이너 여부: 1
capacity: 15
size : 0

C:\Users\click\OneDrive\문서\cppClickseo\x64\Debug\cppClickseo.exe
이 창을 닫으려면 아무 키나 누르세요 ...

vector 클래스 (4/6)

예제 2-6: vector 클래스 조작 함수 -- 동적 배열

(4/4)

// clear: 컨테이너의 모든 요소를 지운다(size() == 0).

// empty: 비어 있는지 여부(true 또는 false)

cout << endl;

v.clear();

cout << "빈 컨테이너 여부: " << **v.empty()** << endl;

cout << "capacity: " << **v.capacity()** << endl;

cout << "size : " << **v.size()** << endl;

cout << endl;

return 0;

}

```
Microsoft Visual Studio 디버그 × + v
최대 요소 수: 4611686018427387903
초기 전체 원소(5): 45 23 36 87 56
중간 전체 원소(15): 99 45 23 36 87 56 0 0 0 0 0 89 56 21 64
정렬된 데이터(15): 0 0 0 0 0 21 23 36 45 56 56 64 87 89 99

빈 컨테이너 여부: 1
capacity: 15
size : 0

C:\Users\click\OneDrive\문서\cppClickseo\x64\Debug\cppClickseo.exe
이 창을 닫으려면 아무 키나 누르세요 ...
```

vector 클래스 (5/6)

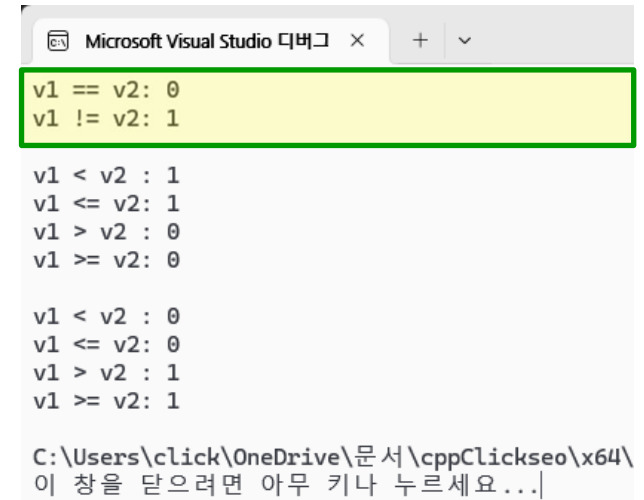
예제 2-7: vector 클래스와 관계 연산자

(1/2)

```
#include <iostream>
#include <vector>      // vector
using namespace std;

int main(void)
{
    vector<int> v1{ 10, 20, 30 }, v2 = { 40, 50 };

    // ==, !=: lhs와 rhs의 내용이 동일한지 확인한다.
    cout << "v1 == v2: " << (v1 == v2) << endl;
    cout << "v1 != v2: " << (v1 != v2) << endl;
    cout << endl;
    // 단, != 연산자는 C++20에서 제거된다(removed in C++20).
```



The screenshot shows the Microsoft Visual Studio debugger window. The top bar indicates 'Microsoft Visual Studio 디버그' with window controls. The main area displays the execution of the program, with a yellow highlight on the first two lines of the output: 'v1 == v2: 0' and 'v1 != v2: 1'. Below these, the results of various comparison operators are shown: 'v1 < v2 : 1', 'v1 <= v2: 1', 'v1 > v2 : 0', 'v1 >= v2: 0', followed by 'v1 < v2 : 0', 'v1 <= v2: 0', 'v1 > v2 : 1', and 'v1 >= v2: 1'. At the bottom, a status bar shows the file path 'C:\Users\click\OneDrive\문서\cppClickseo\x64\' and a message '이 창을 닫으려면 아무 키나 누르세요...'.

vector 클래스 (6/6)

예제 2-8: vector 클래스와 관계 연산자

(2/2)

// <, <=, >, >=: lhs와 rhs의 내용을 사전순으로 비교한다.

```
cout << "v1 < v2 : " << (v1 < v2) << endl;
```

```
cout << "v1 <= v2: " << (v1 <= v2) << endl;
```

```
cout << "v1 > v2 : " << (v1 > v2) << endl;
```

```
cout << "v1 >= v2: " << (v1 >= v2) << endl;
```

```
cout << endl;
```

// 단, <, <=, >, >= 연산자는 C++20에서 제거된다(removed in C++20).

```
v1.swap(v2);
```

```
cout << "v1 < v2 : " << (v1 < v2) << endl;
```

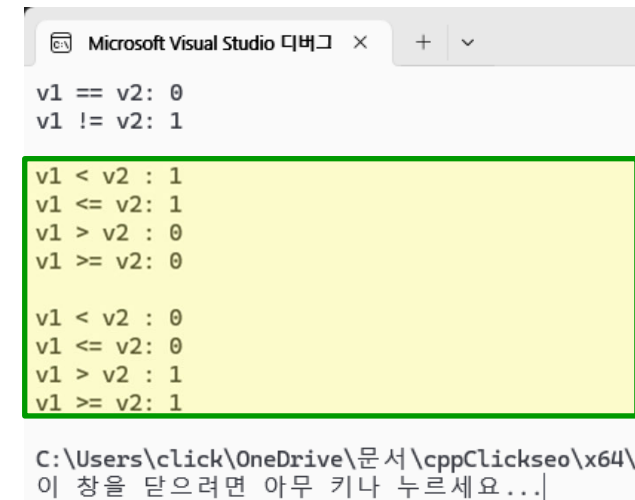
```
cout << "v1 <= v2: " << (v1 <= v2) << endl;
```

```
cout << "v1 > v2 : " << (v1 > v2) << endl;
```

```
cout << "v1 >= v2: " << (v1 >= v2) << endl;
```

```
return 0;
```

```
}
```



```
Microsoft Visual Studio 디버그 x + v
v1 == v2: 0
v1 != v2: 1

v1 < v2 : 1
v1 <= v2: 1
v1 > v2 : 0
v1 >= v2: 0

v1 < v2 : 0
v1 <= v2: 0
v1 > v2 : 1
v1 >= v2: 1

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

선형 리스트

선형 리스트 구현



선형 리스트 (1/4)

- **리스트(List)**

- **목록, 대부분의 목록은 도표(Table) 형태로 표시**

- 추상 자료형 리스트는 이러한 목록 또는 도표를 추상화한 것

이름 리스트	좋아하는 음식 리스트	오늘의 할 일 리스트
홍길동	산채비빔밥	둘레길 산책
이순신	광어회	종이배 만들기
이도	잡채	글쓰기
강감찬	불고기 피자	드라마 보기
...

선형 리스트 (2/4)

● 선형 리스트(Linear List)

○ 순서 리스트(Ordered List)

- 리스트에서 나열한 원소들 간에 순서를 가지고 있는 리스트
- 원소들 간의 논리적인 순서와 물리적인 순서가 같은 구조(순차 자료구조)

이름 리스트		좋아하는 음식 리스트		오늘의 할 일 리스트	
1	홍길동	1	산채비빔밥	1	둘레길 산책
2	이순신	2	광어회	2	종이배 만들기
3	이도	3	잡채	3	글쓰기
4	왕건	4	불고기 피자	4	드라마 보기

선형 리스트 (3/4)

- **선형 리스트:** 원소 삽입

- 선형 리스트에서 원소 삽입

원소 삽입 전

0	1	2	3	4	5	6
10	20	40	50	60	70	

원소 삽입 후

0	1	2	3	4	5	6
10	20	40	50	60	70	

0	1	2	3	4	5	6
10	20	30	40	50	60	70

원소 30 삽입

선형 리스트 (4/4)

- 선형 리스트: 원소 삭제

- 선형 리스트에서 원소 삭제

원소 삭제 전

0	1	2	3	4	5	6
10	20	30	40	50	60	70

원소 삭제 후

0	1	2	3	4	5	6
10	20		40	50	60	70

원소 30 삭제

0	1	2	3	4	5	6
10	20	40	50	60	70	

연결 리스트

단순.원형.이중 연결 리스트

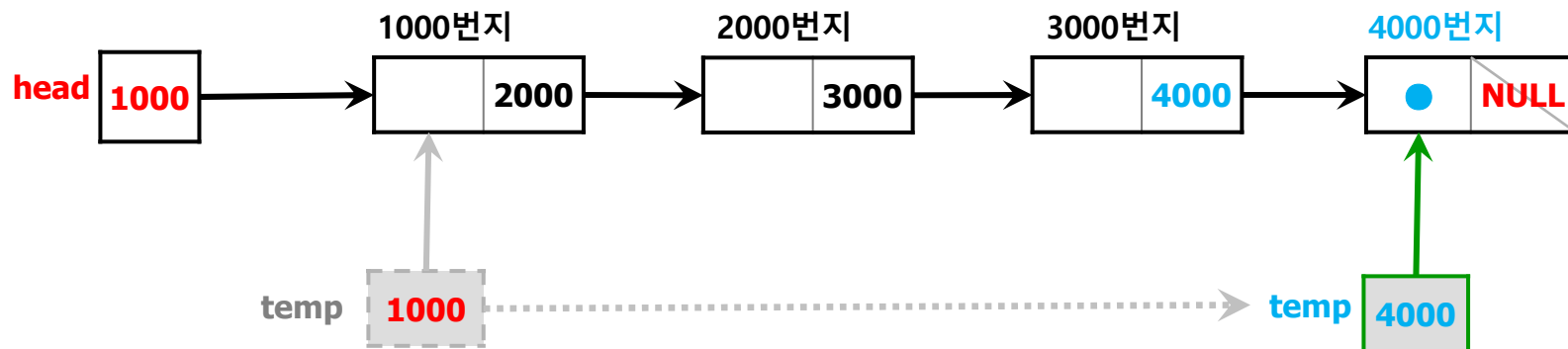


단순 연결 리스트

- 단순 연결 리스트: 탐색 알고리즘

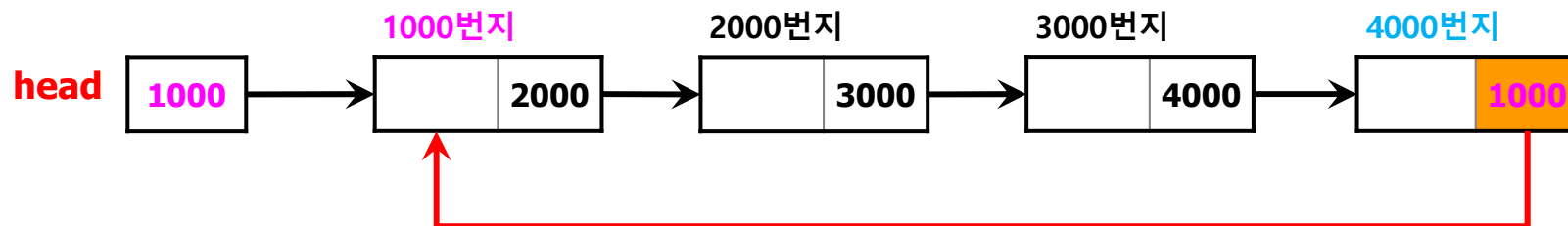
- 리스트에서 조건을 만족하는 데이터를 가진 노드 탐색 알고리즘

```
searchSNode(head, data)
temp ← head;
while (temp != NULL) do
{
    if (temp.data = data) then
        return temp;
    temp ← temp.link;
}
if (temp = NULL) then
    return NULL;
end searchSNode()
```



원형 연결 리스트

- 원형 연결 리스트(Circular linked List)
 - 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 구조를 원형으로 만든 연결 리스트

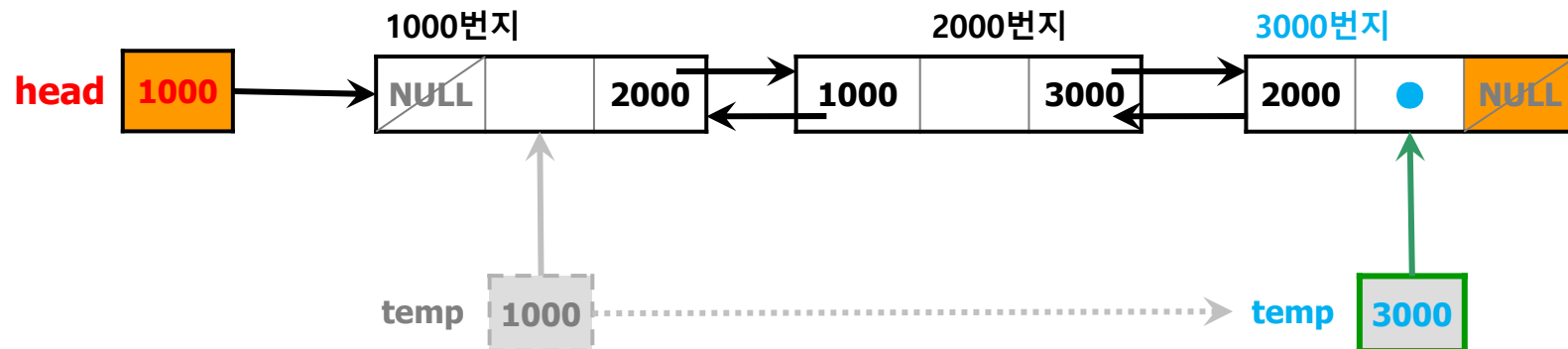


이중 연결 리스트

- 이중 연결 리스트: 탐색 알고리즘

- 리스트에서 조건을 만족하는 데이터를 가진 노드 탐색 알고리즘

```
searchDNode(head, data)
temp ← head;
while (temp != NULL) do
{
    if (temp.data = data) then
        return temp;
    temp ← temp.Rlink;
}
if (temp = NULL) then
    return NULL;
end searchDNode()
```



스택과 큐



- 배열과 리스트

백문이불여일타(百聞而不如一打)

- 스택과 큐

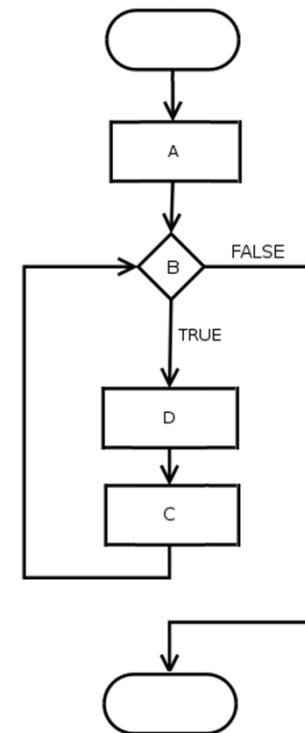
- 스택의 이해

- 큐의 이해

- 트리와 검색 트리

- 해시 테이블

for(A;B;C)
D;



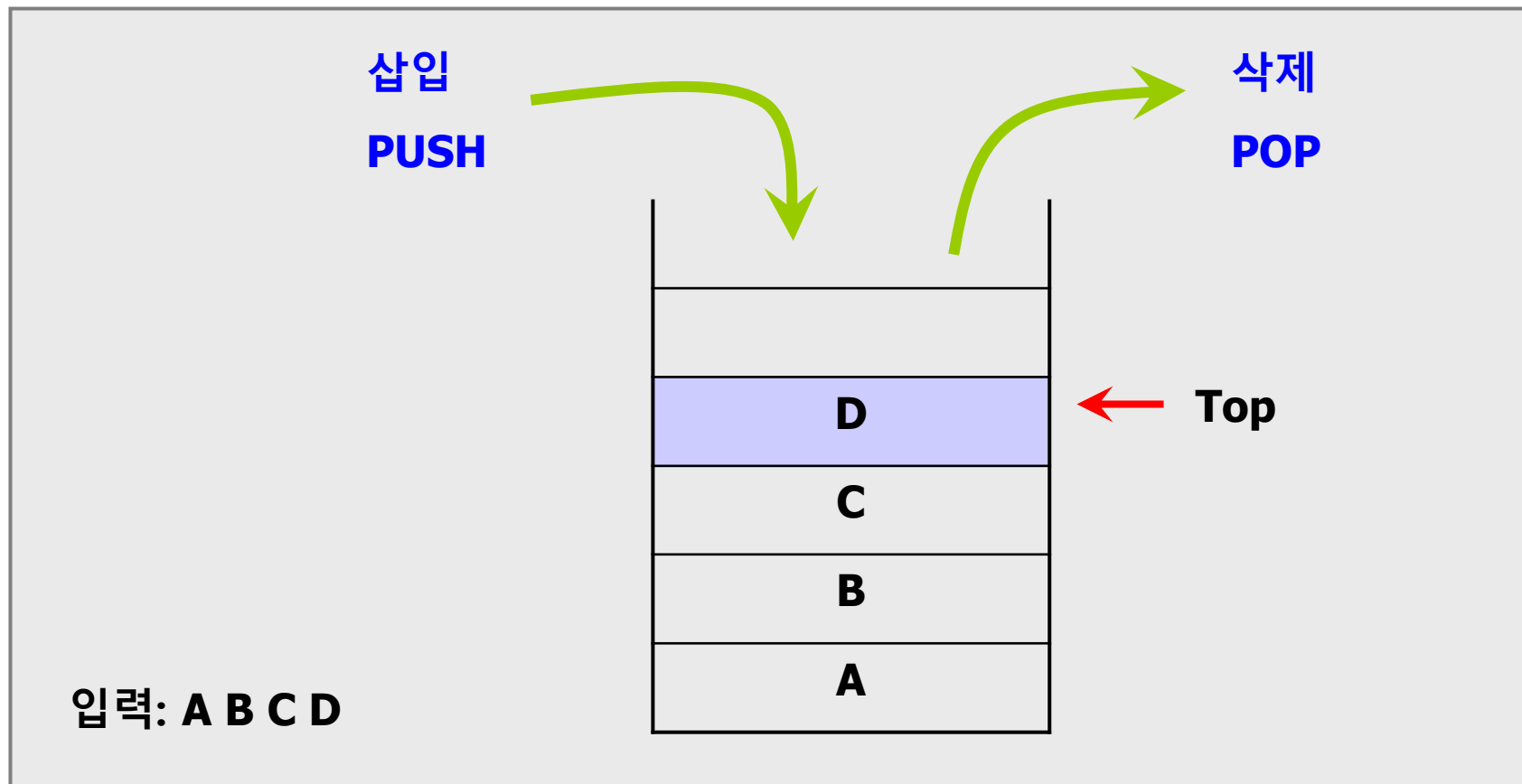
스택과 큐

스택의 이해



스택의 이해

- **스택(Stack)**
 - 후입선출(LIFO, Last-In-First-Out)



Python 내장 클래스: list

예제 2-9: 스택의 이해 -- 파이썬 내장 클래스(list)

| Python

스택(Stack): 후입선출(LIFO, Last-In-First-Out)

S = [] # 빈 리스트 객체 생성

S.append(10)

S.append(20)

S.append(30)

print(f'stack is empty: {len(**S**) == 0}')

print(f'stack size : {len(**S**)}')

while S:

 # **print**(f'top element: {**S.pop**()}')

print(f'top element: {**S**[-1]}')

S.pop()

 IDLE Shell 3.11.2

File Edit Shell Debug Options W

Python 3.11.2 (tags/v3.11
Type "help", "copyright",

>>>

===== RESTART:

stack is empty: False

stack size : 3

top element: 30

top element: 20

top element: 10

>>>

C++ STL: stack 클래스 (1/2)

- **stack** 클래스

- 스택, LIFO(Last in First out)

```
// C++ STL: <stack>
#include <stack>
using namespace std;

stack<DataType> stackName           // 빈 스택 생성

void          push(const value_type& val);  // 스택에 데이터 항목 추가
void          pop();                        // 스택에 데이터 항목 삭제
value_type&   top();                       // 스택의 데이터 항목 반환(top)
bool          empty() const;               // 스택이 비어 있는지 여부 확인
size_type     size() const;                // 스택의 크기 반환
swap(stack1, stack2)                  // 스택 SWAP
```

C++ STL: stack 클래스 (2/2)

예제 2-10: 스택의 이해 -- C++ STL(stack class)

| C++

```
#include <iostream>
#include <stack>
using namespace std;
```

```
int main(void)
{
```

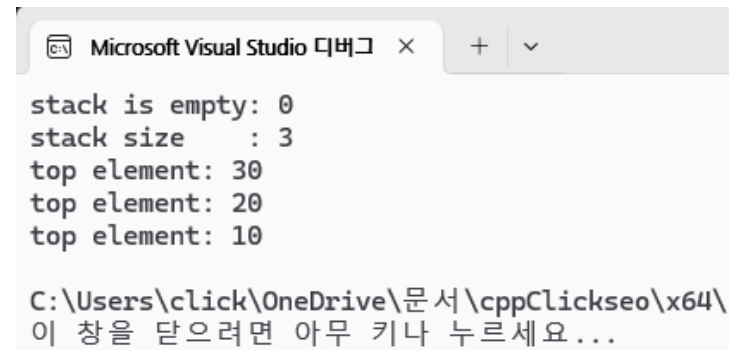
```
    // 빈 스택 생성
    stack<int> S;
```

```
    S.push(10);
    S.push(20);
    S.push(30);
```

```
    cout << "stack is empty: " << S.empty() << endl;
    cout << "stack size      : " << S.size() << endl;
```

```
    while (!S.empty()) {
        cout << "top element: " << S.top() << endl;
        S.pop();
    }
    return 0;
```

```
}
```



```
Microsoft Visual Studio 디버그 x + v

stack is empty: 0
stack size      : 3
top element: 30
top element: 20
top element: 10

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

스택과 큐

큐의 이해

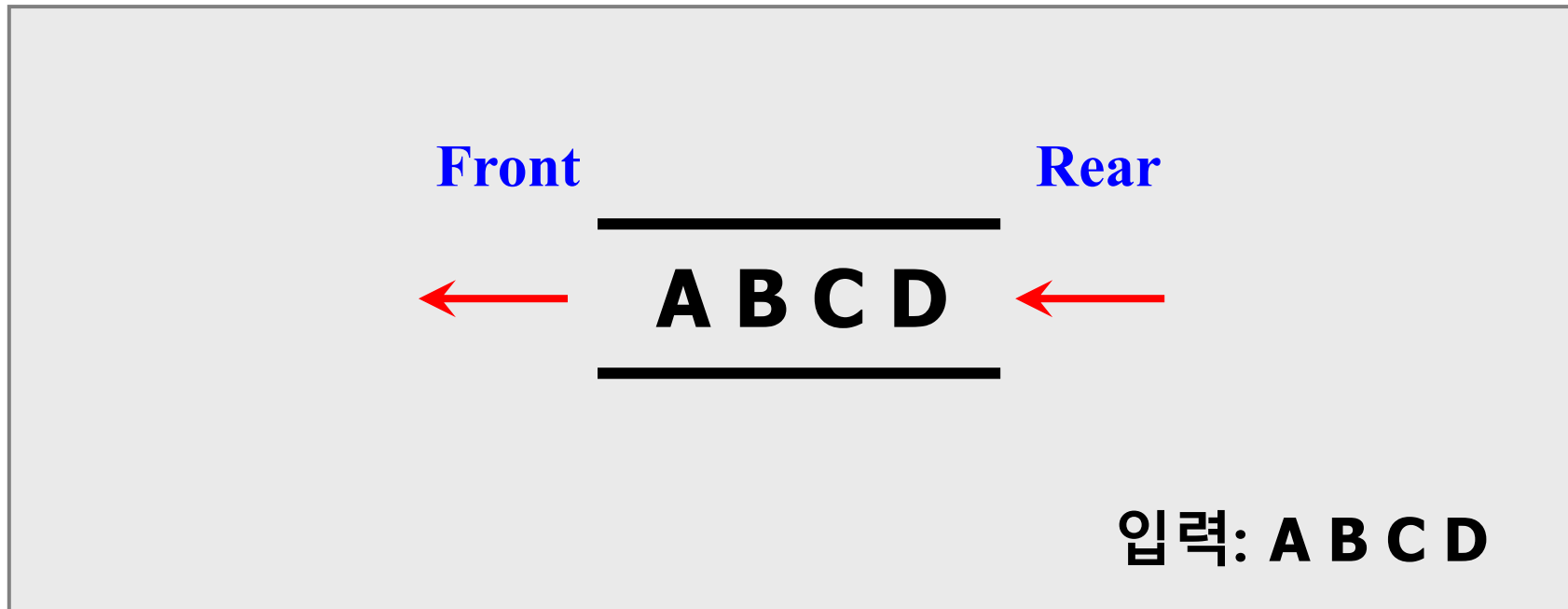


큐의 이해

- 큐(Queue)

- 선입선출(FIFO, First-In First-Out)

- 리스트의 한쪽 끝에서 삽입 작업이 이루어지고 반대쪽 끝에서 삭제 작업이 이루어져서 삽입된 순서대로 삭제되는 구조



Python 내장 클래스: list

예제 2-11: 큐의 이해 -- 파이썬 내장 클래스(list)

| Python

큐(Queue): 선입선출(FIFO, First-In First-Out)

Q = [] # 빈 리스트 객체 생성

Q.append(10)

Q.append(20)

Q.append(30)

print(f'queue is empty: {len(Q) == 0}')

print(f'queue size : {len(Q)}')

while Q:

print(f'front element: {S.pop(0)}')

print(f'front element: {Q[0]}')

Q.pop(0)

IDLE Shell 3.11.2

File Edit Shell Debug Options W

Python 3.11.2 (tags/v3.11.
Type "help", "copyright",

>>>

===== RESTART:

queue is empty: False

queue size : 3

front element: 10

front element: 20

front element: 30

>>>

C++ STL: queue 클래스 (1/2)

- **queue** 클래스

- 큐, FIFO(First in First out)

```
// C++ STL : <queue>
#include <queue>
using namespace std;

queue<DataType> queueName           // 빈 큐 생성

void          push(const value_type& val); // 큐에 데이터 추가
void          pop();                      // 큐에 데이터 삭제
value_type&   front();                    // 큐의 첫 번째 원소 반환
value_type&   back();                     // 큐의 마지막 원소 반환
bool          empty() const;              // 큐가 비어 있는지 여부 확인
size_type     size() const;               // 큐의 크기 반환
```

C++ STL: queue 클래스 (2/2)

예제 2-12: 큐의 이해 -- C++ STL(queue class)

| C++

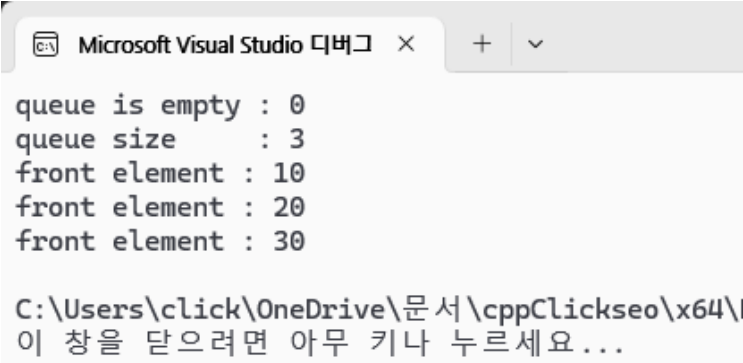
```
#include <iostream>
#include <queue>
using namespace std;

int main(void)
{
    queue<int>    Q;

    Q.push(10);
    Q.push(20);
    Q.push(30);

    cout << "queue is empty : " << Q.empty() << endl;
    cout << "queue size      : " << Q.size() << endl;

    while (!Q.empty()) {
        cout << "front element : " << Q.front() << endl;
        Q.pop();
    }
    return 0;
}
```



Microsoft Visual Studio 디버그 × + ▾

```
queue is empty : 0
queue size      : 3
front element   : 10
front element   : 20
front element   : 30
```

C:\Users\click\OneDrive\문서\cppClickseo\x64\이 창을 닫으려면 아무 키나 누르세요...

트리와 검색 트리



- 배열과 리스트

- 스택과 큐

- 트리와 검색 트리

- 이진 트리

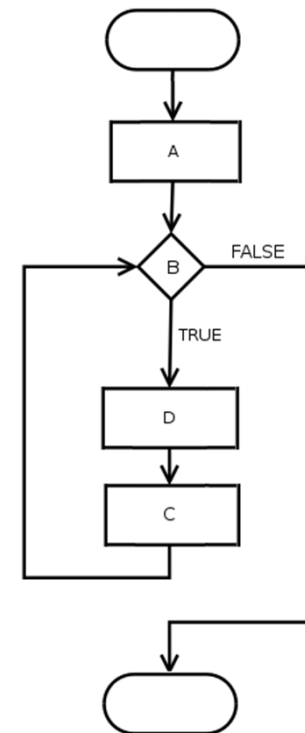
- 우선 순위 큐와 힙

- 검색 트리

- 해시 테이블

백문이불여일타(百聞而不如一打)

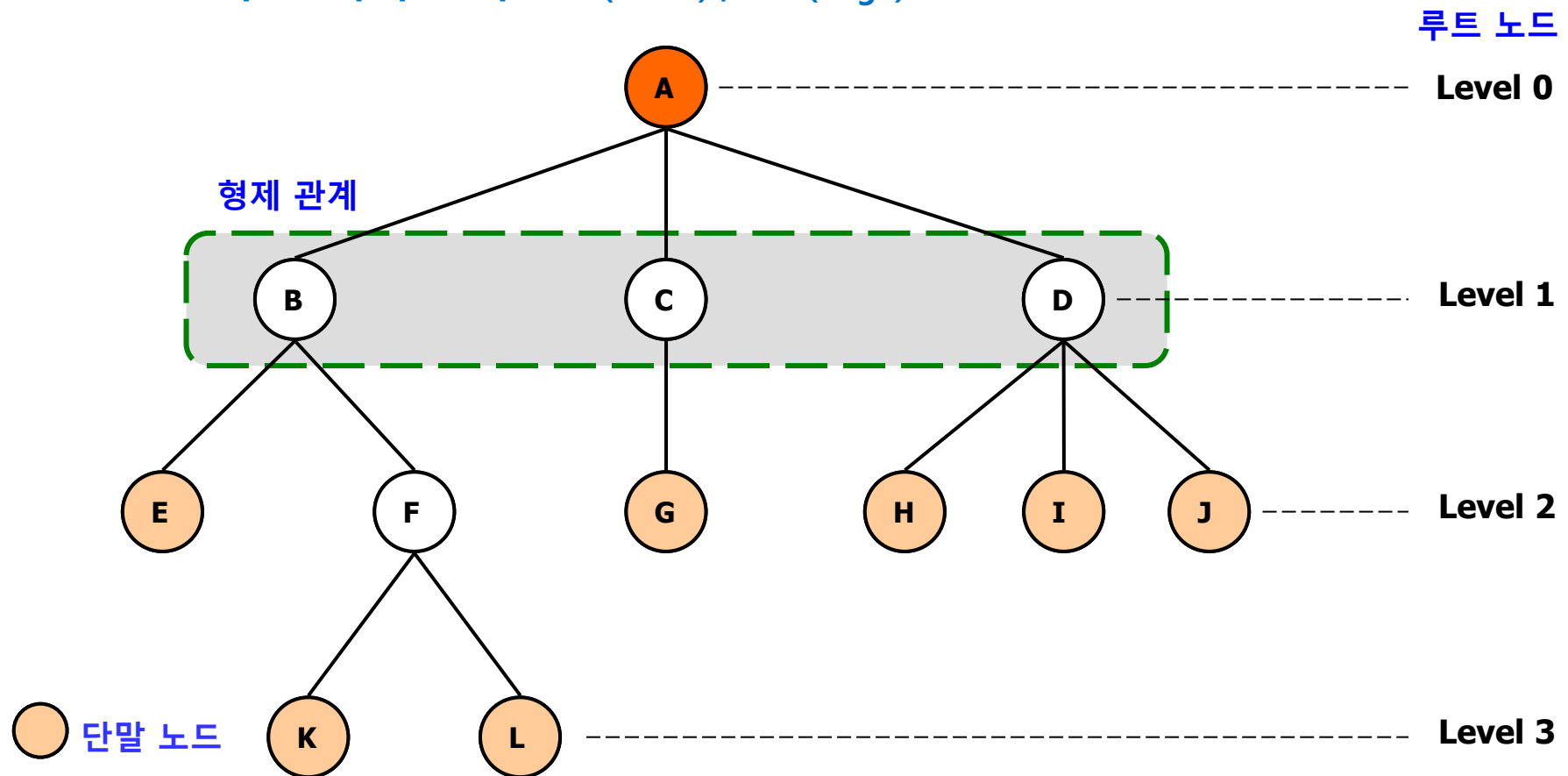
for(A;B;C)
D;



트리의 이해

- 트리 구조

- 부모-자식 관계: 노드(Node) , 간선(Edge)



트리와 검색 트리

이진 트리

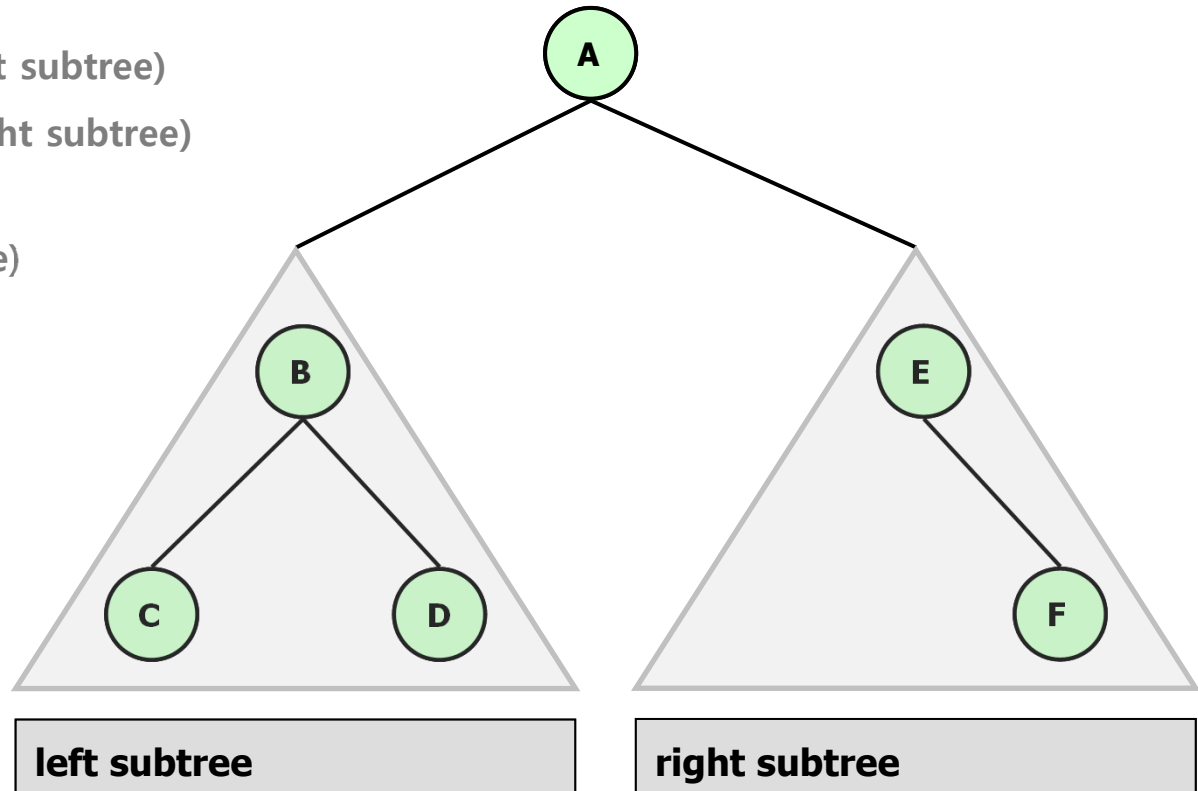


이진 트리 (1/3)

- **이진 트리(Binary Tree)**

- 최대 두 개까지의 자식 노드를 가질 수 있는 트리

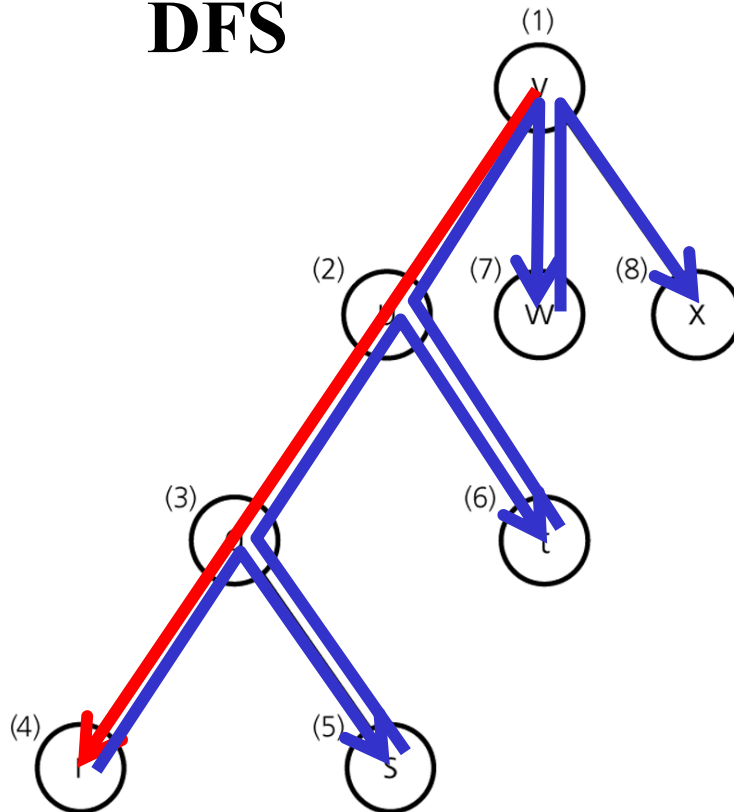
- 하나의 노드는 0, 1, 혹은 2개의 서브 트리를 가질 수 있다.
- 좌 서브 트리(left subtree)
- 우 서브 트리(right subtree)
- 널 트리(null tree)



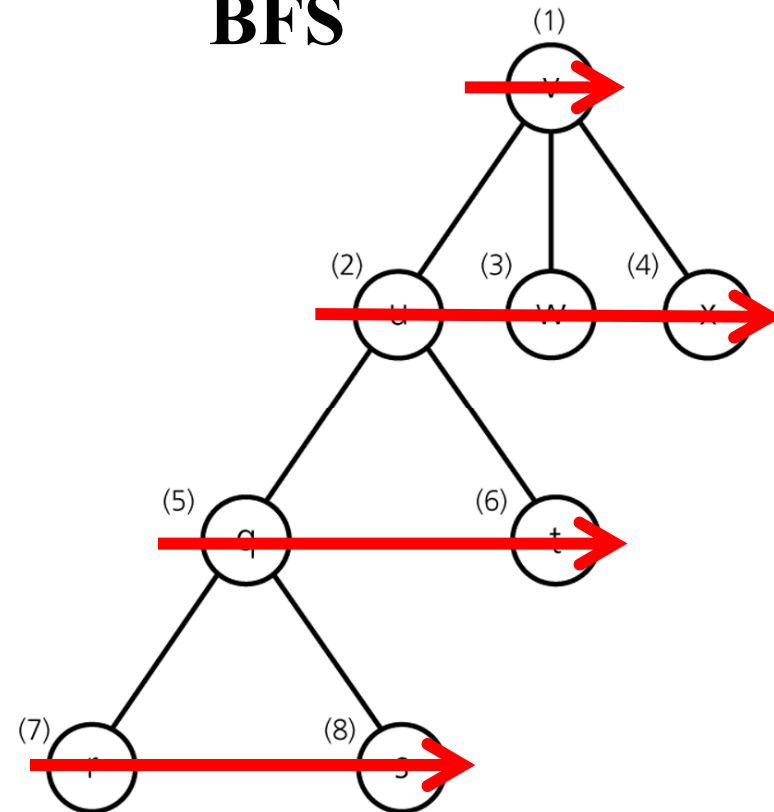
이진 트리: 순회 (2/3)

- (이진) 트리 순회(Traversal)
 - 깊이 우선 순회(DFS)와 너비 우선 순회(BFS)

DFS



BFS



이진 트리: 순회 (3/3)

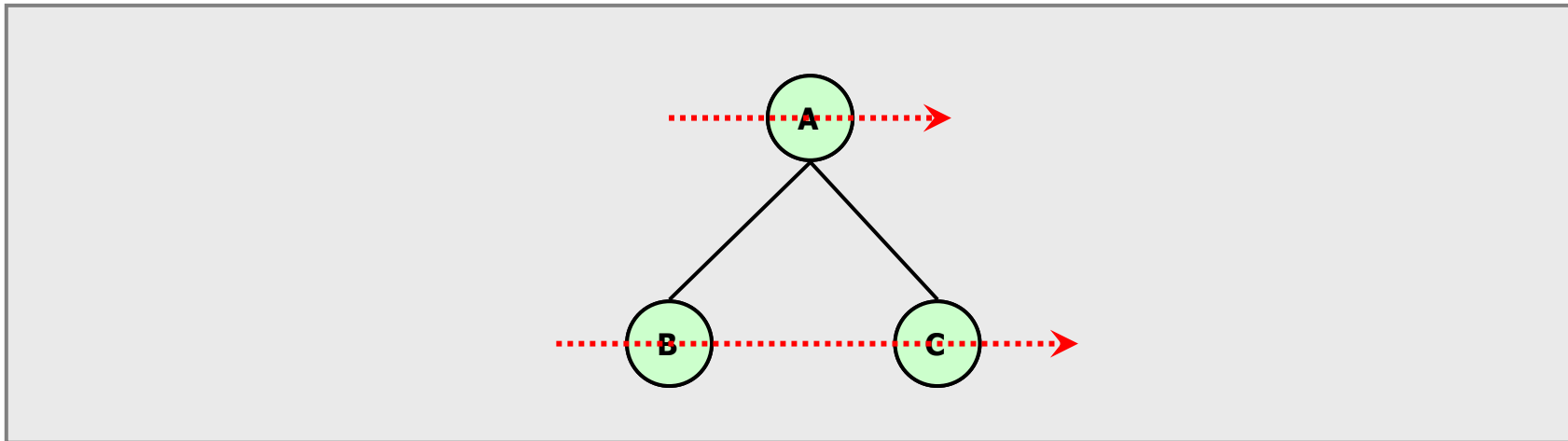
- **이진 트리 순회: DFS, BFS**

- **깊이 우선 순회: 스택을 이용하여 구현**

- 전위 순회(preorder traversal)
 - 중위 순회(inorder traversal)
 - 후위 순회(postorder traversal)

- **너비 우선 순회: 큐를 이용하여 구현**

- 다음 레벨의 노드들을 처리하기 전에 노드의 자식 모두를 처리



트리와 검색 트리

우선 순위 큐와 힙



우선 순위 큐와 힙 (1/3)

- **힙(Heap)**

- **우선 순위 큐를 구현하는 가장 기본적인 자료구조**

- 힙은 다음 두 조건을 만족해야 한다
 1. 완전 이진 트리
 2. 모든 노드는 값을 갖고, 자식 노드(들) 값보다 크거나 같다.

- **우선 순위 큐(Priority Queue)**

- 가장 높은 우선순위를 가진 항목에 접근, 삭제와 임의의 우선순위를 가진 항목을 삽입을 지원하는 자료구조
- 스택이나 큐도 일종의 우선 순위 큐
 - 스택: 가장 마지막으로 삽입된 항목이 가장 높은 우선순위를 가진다.
 - » 따라서 최근 시간일수록 높은 우선순위를 부여한다.
 - 큐: 먼저 삽입된 항목이 우선순위가 더 높다
 - » 따라서 이른 시간일수록 더 높은 우선순위를 부여한다.
 - 삽입되는 항목이 임의의 우선순위를 가지면 스택이나 큐는 새 항목이 삽입될 때마다 저장되어 있는 항목들을 우선순위에 따라 정렬해야 하는 문제점이 있음.

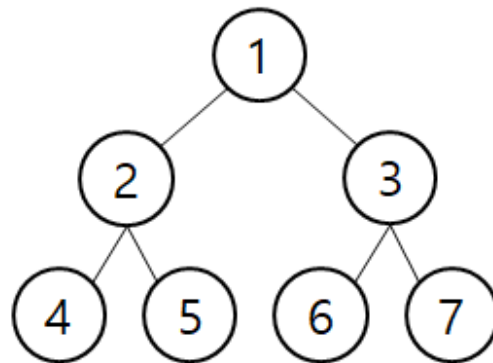
우선 순위 큐와 힙: 최소 힙 (2/3)

● 최소 힙(Minimum Heap)

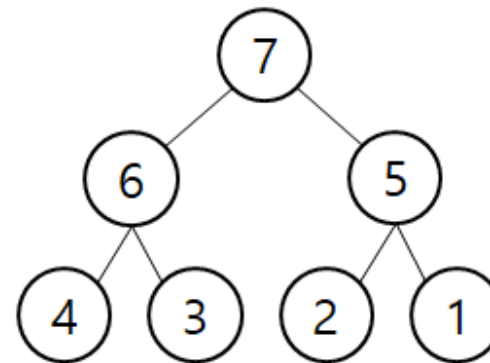
○ 키 값이 작을수록 높은 우선순위

- 최소 힙의 루트에는 항상 가장 작은 키가 저장된다.
 - 부모에 저장된 키가 자식의 키보다 작다는 규칙
 - 루트는 $a[1]$ 에 있으므로, $O(1)$ 시간에 min 키를 가진 노드 접근

○ 최대 힙: 키 값이 클수록 더 높은 우선순위



최소힙

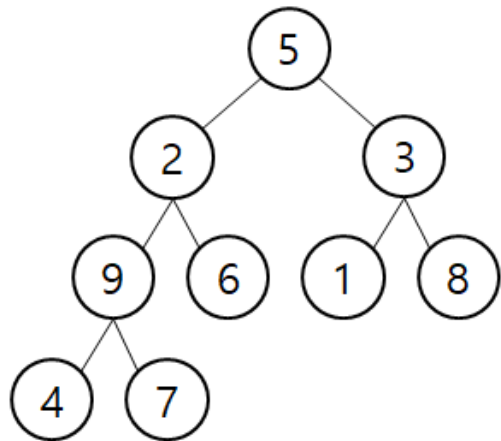


최대힙

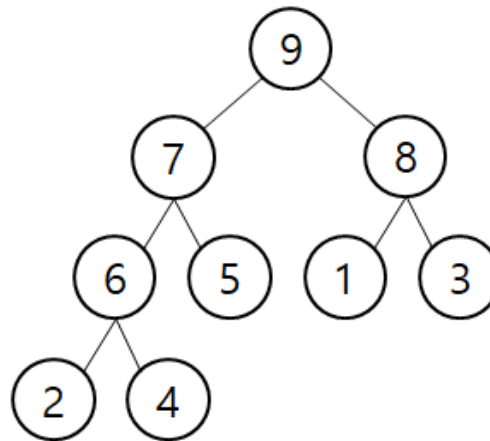
우선 순위 큐와 힙: 힙 정렬 (3/3)

- **힙 정렬**(Heap Sort)

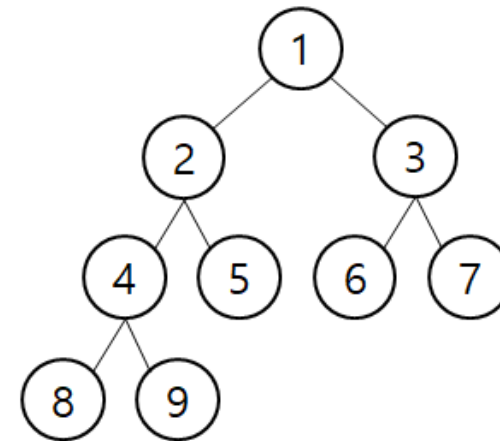
- 힙을 이용한 정렬 과정



리스트를 완전 이진 트리로 표현



최대힙



정렬 완료

트리와 검색 트리

검색 트리



검색 트리

- **검색 트리(Binary Search Tree)**

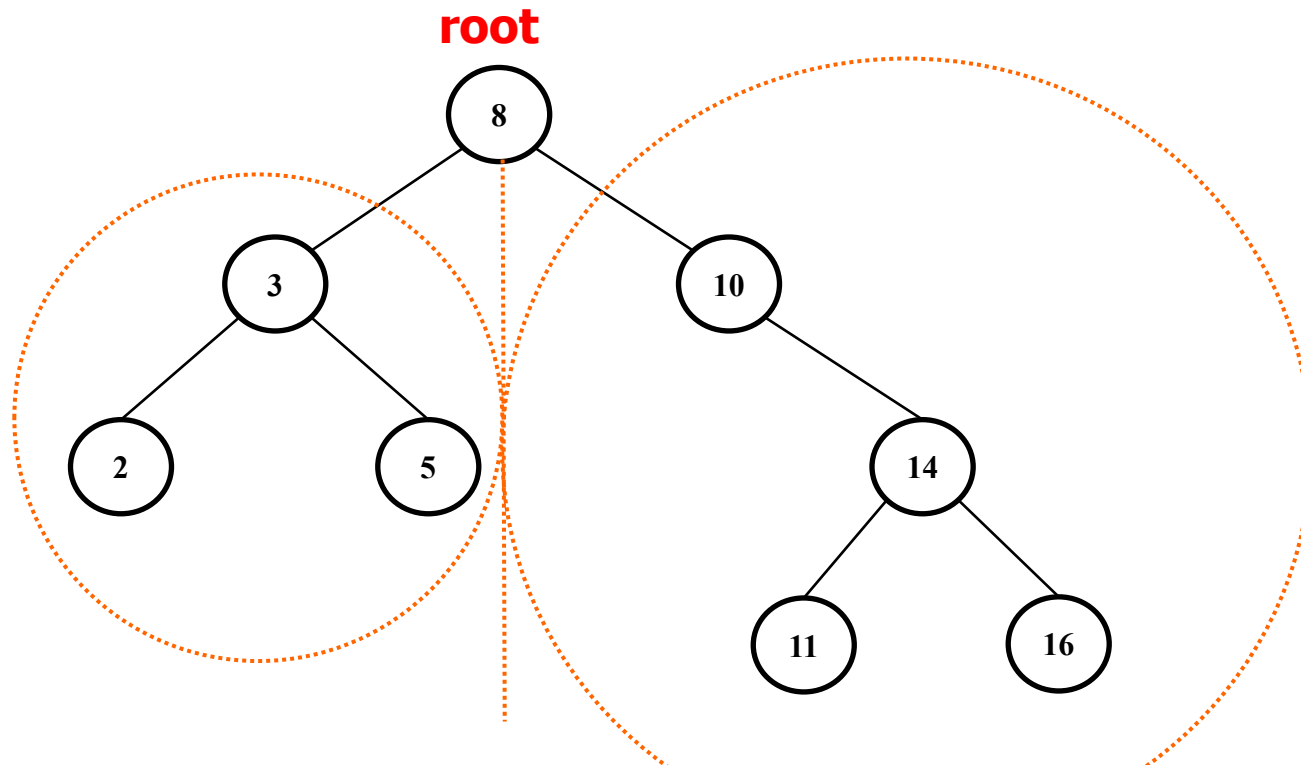
- 검색 트리: 이진 검색 트리, 다진 검색 트리



[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

이진 검색 트리 (1/3)

- **이진 검색 트리**(Binary Search Tree)
 - 모든 노드는 서로 다른 키를 갖는다(유일한 키 값).
 - 각 노드는 최대 2개의 자식을 갖는다.



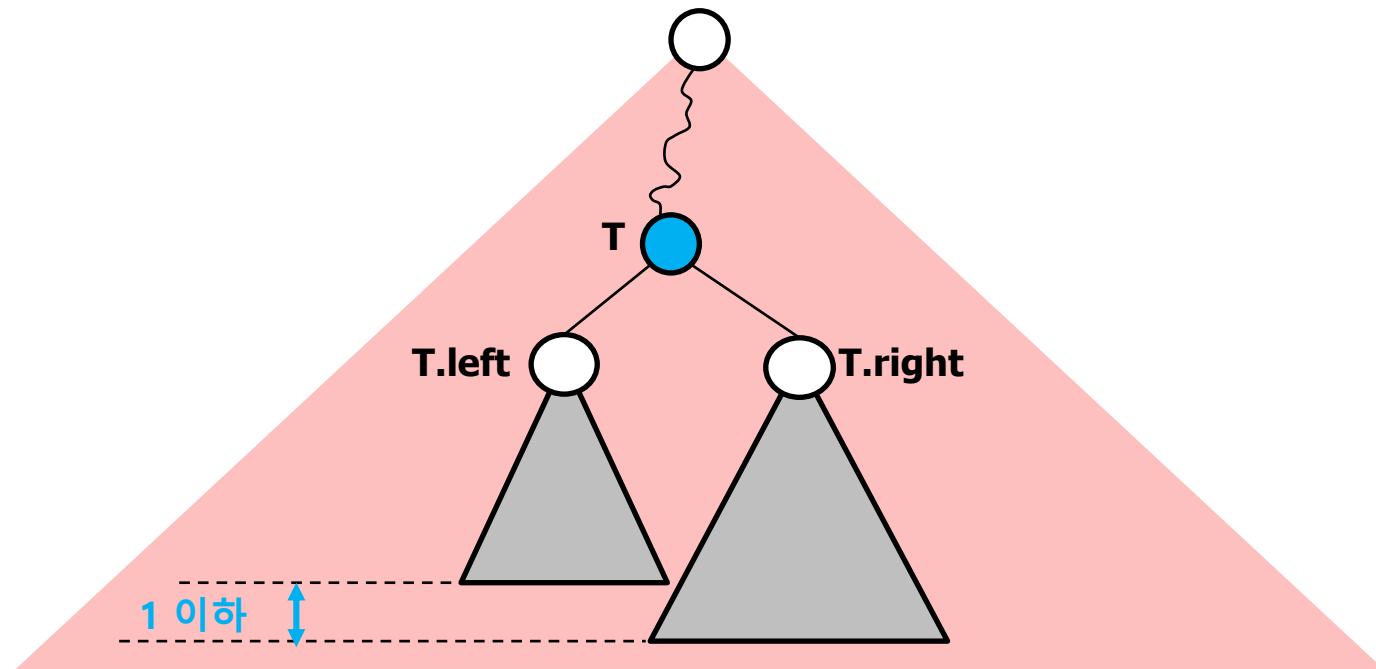
왼쪽 서브 트리의 키 값 < 루트의 키 값 < 오른쪽 서브 트리의 키 값

이진 검색 트리: AVL 트리 (2/3)

- **AVL 트리**

- 전체 트리의 구조가 균형이 맞도록 하는 트리

- 모든 노드에 대해 좌 서브 트리의 높이(깊이)와 우 서브 트리의 높이의 차이가 1을 넘지 않는다(즉, 트리 구조가 한쪽으로 쏠리는 것을 막을 수 있다).

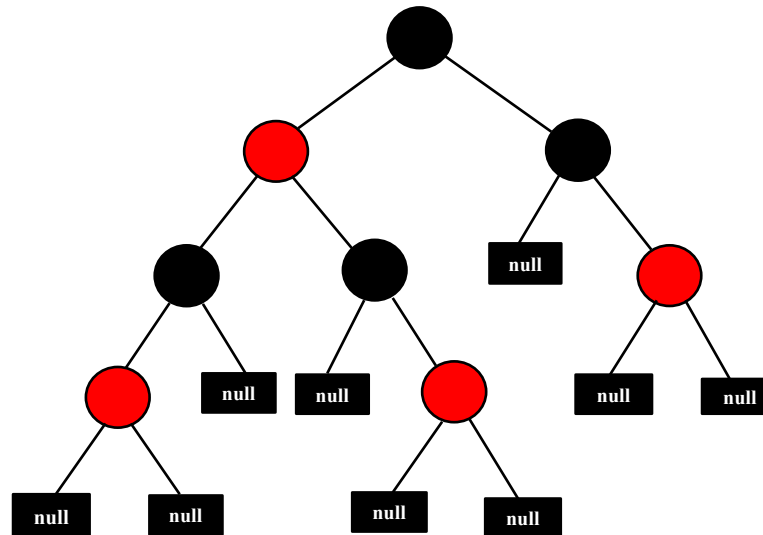


이진 검색 트리: 레드-블랙 트리 (3/3)

● 레드-블랙 트리: 특성

○ 레드-블랙 트리의 특성

- 모든 null 자리에 단말 노드((leaf node)를 둔다.
 - RB-Tree에서 단말 노드는 이 null을 말한다.
- 모든 노드는 Red 또는 Black의 색을 갖는다.
 - 루트와 모든 단말 노드는 블랙이다.
 - 임의의 단말 노드에 이르는 경로 상에 레드 노드 두 개가 연속으로 출현하지 못한다.
 - 임의의 단말 노드에 이르는 경로에서 만나는 블랙 노드의 수(black height)는 모두 같다.



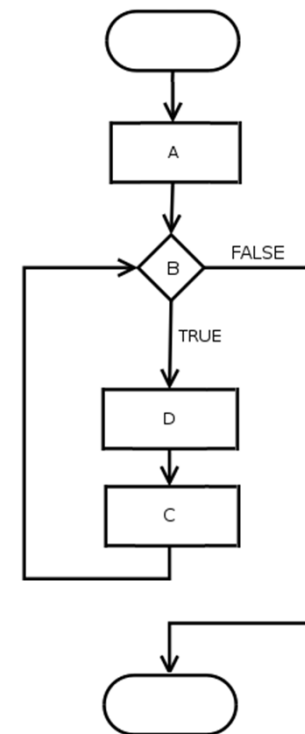
해시 테이블



백문이불여일타(百聞而不如一打)

- 배열과 리스트
- 스택과 큐
- 트리와 검색 트리
- 해시 테이블
 - 충돌 해결

for(A;B;C)
D;



해시 테이블 (1/2)

입력 : 25, 13, 16, 15, 7

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

그림 12-1 해시 테이블 예

● 해시 테이블(Hash Table)

○ 키를 변환하여 배열의 인덱스로 사용한다.

- $\Theta(1)$: 아주 빠른 검색, 삽입, 삭제 작업을 제공한다.
 - 자신의 키값에 의해 위치가 결정된다.
 - 다른 키값과의 상대적인 크기에 의해 위치가 결정되지 않는다.
- 키를 배열의 인덱스로 그대로 사용하면 메모리 낭비가 심해질 수 있다.

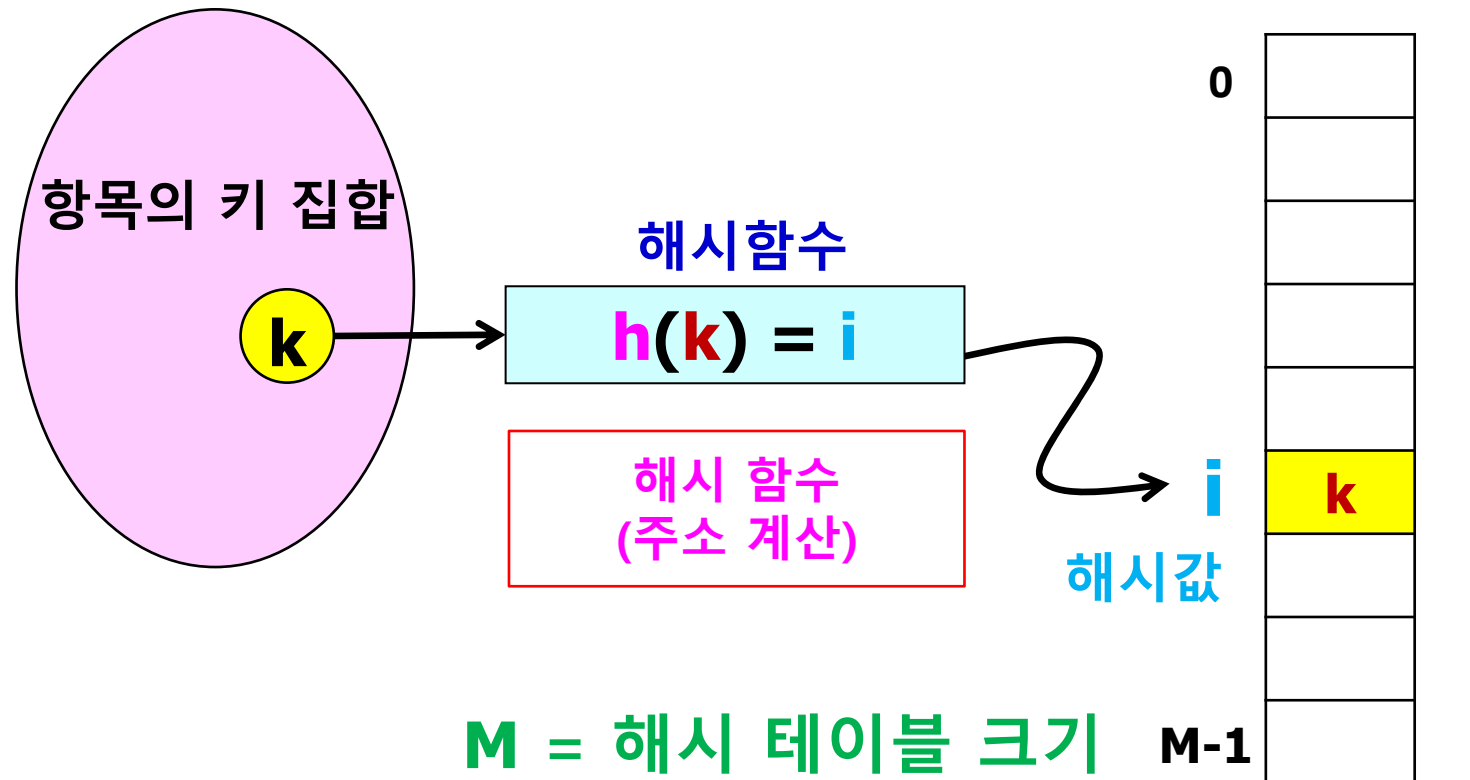
○ 해싱(Hashing)

- 키를 간단한 함수를 사용해 변환한 값을 배열의 인덱스로 이용하여 항목을 저장하는 것
- 해시 함수(Hash Function): 해싱에 사용되는 함수
- 해시 값(Hash value) 또는 해시주소: 해시함수가 계산한 값
- 해시 테이블(Hash Table): 항목이 해시 값에 따라 저장되는 배열

해시 테이블 (2/2)

- 해시 테이블: 주소 계산

- 주소 계산



해시 테이블

충돌 해결



충돌 해결 (1/3)

● 충돌 해결(Collision Resolution)

○ 충돌 해결

충돌(Collision) :

어떤 키값으로 도출된 주소에 이미 다른 키가 자리함

충돌 해결

- 일련의 해시 함수를 생성한다.
- $h_0(x)(=h(x)), h_1(x), h_2(x), h_3(x), \dots$
- 해시 테이블에서 핵심적인 부분

$$h(224) = 224 \% 101 = 22$$

table[22]가
이미 점유됨

table[]

0	
	⋮
22	123
	⋮
100	

$$\text{해시 함수: } h(x) = x \% 101$$

충돌 해결: 개방형 주소 방식 (2/3)

● 개방형 주소 방식(Open Addressing Methods)

○ 해시테이블 전체를 열린 공간으로 가정하고 충돌된 키를 일정한 방식에 따라서 찾아낸 empty 원소에 저장한다.

- 선형 조사(Linear Probing)
- 이차원 조사(Quadratic Probing)
- 무작위 조사(Random Probing)
- 이중 해싱(Double Hashing)

개방 주소 방법(Open Addressing)

: 주어진 배열 안에서 해결

선형 조사(Linear Probing)

$$h_i(x) = (h_0(x) + ai + b) \% m$$

이차원 조사(Quadratic Probing)

$$h_i(x) = (h_0(x) + ai^2 + bi + c) \% m$$

이중 해싱(Double Hashing)

$$h_i(x) = (h_0(x) + i \cdot f(x)) \% m$$

$f(x)$: 보조 해시 함수

m : 해시 테이블 크기, $\%$: 나머지 연산

충돌 해결: 폐쇄 주소 방식 (3/3)

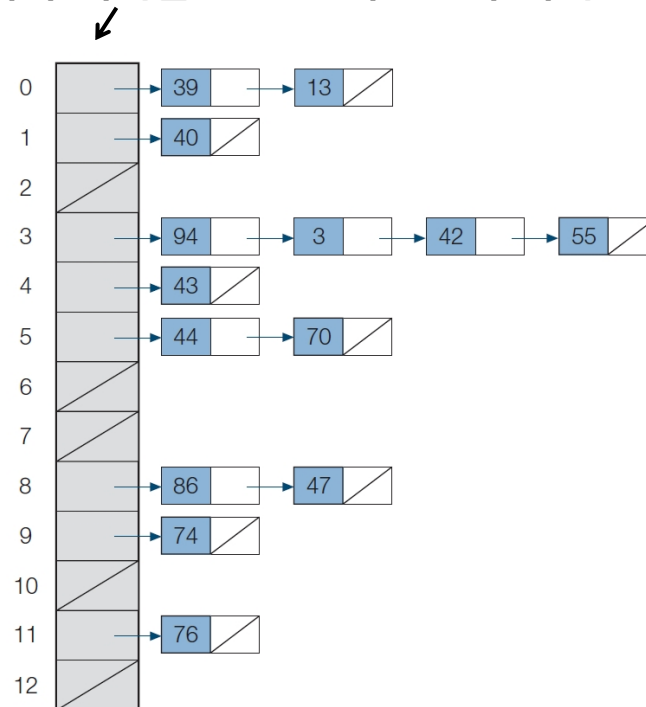
- **폐쇄 주소 방식**(Closed Addressing Methods)

- 키에 대한 해시 값에 대응되는 곳에만 키를 저장한다.

- 충돌이 발생한 키들은 한 위치에 모여 저장한다.

- 가장 대표적인 방법: **체이닝**(Chaining)

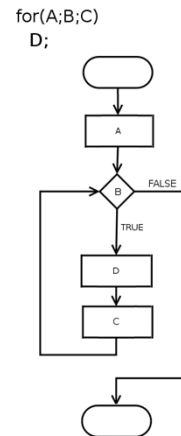
해시 테이블은 연결 리스트의 헤더 노드를 참조한다



직접 충돌이 일어나지 않은
키들끼리는 간섭하지 않는다.

참고문헌

- [1] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [2] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛아카데미, 2022.
- [3] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.
- [5] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [6] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2024 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2024 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2024 of viewing the site, <https://programmers.co.kr/>.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

