3. C 프로그래밍 복습 2

한국외국어대학교
고 석 훈

목차

3.5 동적 메모리(dynamic memory)

- 동적 메모리 할당(dynamic memory allocation)
  - 앞의 예에서는 이미 만들어진 변수의 메모리 주소를 포인터 변수가 가리키는 단순한 방식만 사용했다.
  - 별도의 변수 없이 동적(dynamic)으로 새로운 메모리 영역을 할당하여 그 주소를 포인터 변수로 가리켜 사용하는 방법을 동적 메모리 할당이라 한다.

malloc/free 함수

- 동적 메모리를 할당하는 malloc 함수
  - `void *malloc(size_t n)`
    - n byte 크기의 메모리를 할당하고, 시작주소를 리턴 한다.

- 동적 메모리를 반환하는 free 함수
  - `void free(void* p)`
    - 동적 할당 받은 메모리 p를 시스템에 반납한다.

```
int  *p;
p = (int*)malloc(sizeof(int)*n);
. . .
free(p);
```

예제 3-7: 동적 메모리 할당

```
#include <stdio.h>
#include <stdlib.h>

typedef struct name_card {
    char   name[20];
    char   phone[20];
    int    birthday;
} CardType;

void printCard(CardType c) {
    printf("%s, %s, %d\n", c.name, c.phone, c.birthday);
}

void main() {
    CardType c1 = { "홍길동", "010-111-1111", 710101 };
    CardType *pc2, *pc3;

    pc2 = &c1;
    pc3 = (CardType*)malloc(sizeof(CardType));
    *pc3 = c1;

    strcpy(c1.name, "임꺽정");
    strcpy(pc2->phone, "010-222-2222");
    strcpy(pc3->phone, "010-333-3333");

    printCard(c1);
    printCard(*pc2);
    printCard(*pc3);

    free(pc3);
}
```

```
임꺽정, 010-222-2222, 710101
임꺽정, 010-222-2222, 710101
홍길동, 010-333-3333, 710101
```

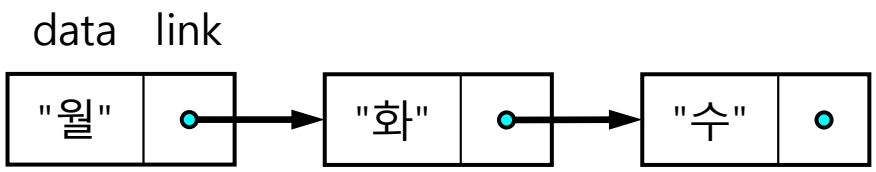
메모리 구성

- 코드(Code) 영역
  - 프로그램 코드가 저장됨
- 데이터(Data) 영역
  - 상수, 전역변수와 static 변수가 저장되는 영역
  - 프로그램이 끝날 때까지 유지된다.
- 스택(Stack) 영역
  - 지역변수와 매개변수가 저장되는 영역
  - 함수가 호출 될 때 생성되며(push), 리턴 될 때 제거된다.(pop)
- 힙(Heap) 영역
  - 동적 메모리 할당에 사용되는 영역
  - malloc에 의해 할당되며, free에 의해 반납된다.

3.6 연결 리스트(Linked List)

- 연결 리스트(linked list)
  - 리스트를 연결 자료구조로 표현한 구조

- 노드(node)
  - 연결 자료구조에서 하나의 원소를 표현하기 위한 단위 구조
  - 데이터 필드와 링크 필드로 구성




```
typedef struct _Node {
   char data[20];
   struct _Node*  link;
} Node;
```

예제 3.8: 연결 리스트 생성

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _Node {
   char data[20];
   struct _Node*  link;
} Node;

Node *newNode(char data[]) {
   Node *p = (Node*)malloc(sizeof(Node));
   strcpy(p->data, data);
   p->link = NULL;
   return p;
}

Node *addFirstNode(Node *list, Node *p) {
   p->link = list;
   return p;
}

void main() {
   Node *list = NULL;

   list = addFirstNode(list, newNode("Wed"));
   list = addFirstNode(list, newNode("Tue"));
   list = addFirstNode(list, newNode("Mon"));
}
```

예제 3.9: 리스트 출력

- printList 함수를 만들어 보자.

```
. . .

void printList(Node *list) { . . . }

void main() {
   Node *list = NULL;

   printList(list);
   list = addFirstNode(list, newNode("Wed"));
   printList(list);
   list = addFirstNode(list, newNode("Tue"));
   printList(list);
   list = addFirstNode(list, newNode("Mon"));
   printList(list);
}
```

```
NULL list
[Wed]
[Tue]->[Wed]
[Mon]->[Tue]->[Wed]
```

예제 3.10: 마지막 노드 추가

- addLastNode 함수를 만들어 보자.

```
. . .

Node *addLastNode(Node *list, Node *p) { . . . }

void main() {
   Node *list = NULL;

   printList(list);
   list = addFirstNode(list, newNode("Wed"));
   printList(list);
   list = addFirstNode(list, newNode("Tue"));
   printList(list);
   list = addFirstNode(list, newNode("Mon"));
   printList(list);

   list = addLastNode(list, newNode("Thu"));
   printList(list);
   list = addLastNode(list, newNode("Fri"));
   printList(list);
}
```

```
NULL list
[Wed]
[Tue]->[Wed]
[Mon]->[Tue]->[Wed]
[Mon]->[Tue]->[Wed]->[Thu]
[Mon]->[Tue]->[Wed]->[Thu]->[Fri]
```

예제 3.11: 리스트 제거, freeList

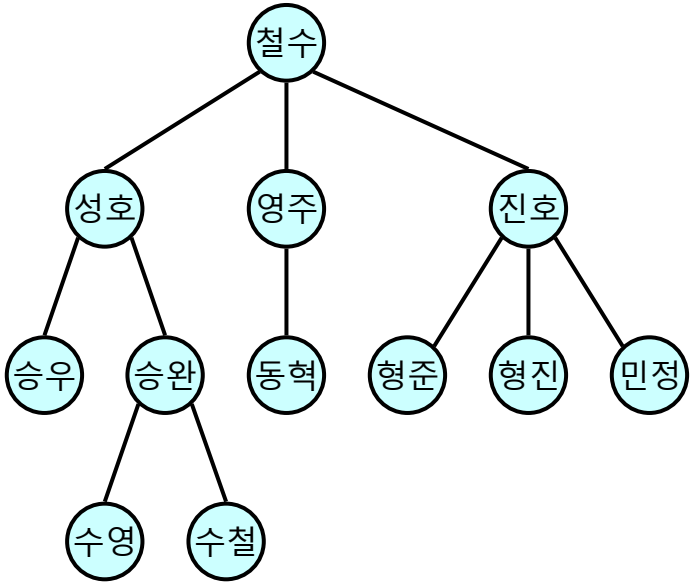
```
. . .

void freeList(Node *list) {
   Node *p;

   while (list != NULL) {
      p = list;
      list = list->link;
      free(p);
   }
}

void main() {
   Node *list = NULL;

   printList(list);
   list = addFirstNode(list, newNode("Wed"));
   printList(list);
   list = addFirstNode(list, newNode("Tue"));
   printList(list);
   list = addFirstNode(list, newNode("Mon"));
   printList(list);

   list = addLastNode(list, newNode("Thu"));
   printList(list);
   list = addLastNode(list, newNode("Fri"));
   printList(list);

   freeList(list);
}
```

```
NULL list
[Wed]
[Tue]->[Wed]
[Mon]->[Tue]->[Wed]
[Mon]->[Tue]->[Wed]->[Thu]
[Mon]->[Tue]->[Wed]->[Thu]->[Fri]
```

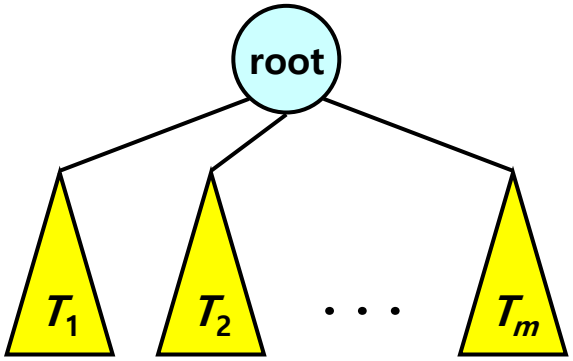
3.7 트리(Tree) 자료구조

- 트리(tree)
  - 원소들 간에 1:多 관계를 가지는 비선형 자료구조
  - 원소들 간에 계층관계를 가지는 계층형 자료구조
  - 상위 원소에서 하위 원소로 내려가면서 확장되는 트리(나무)모양의 구조

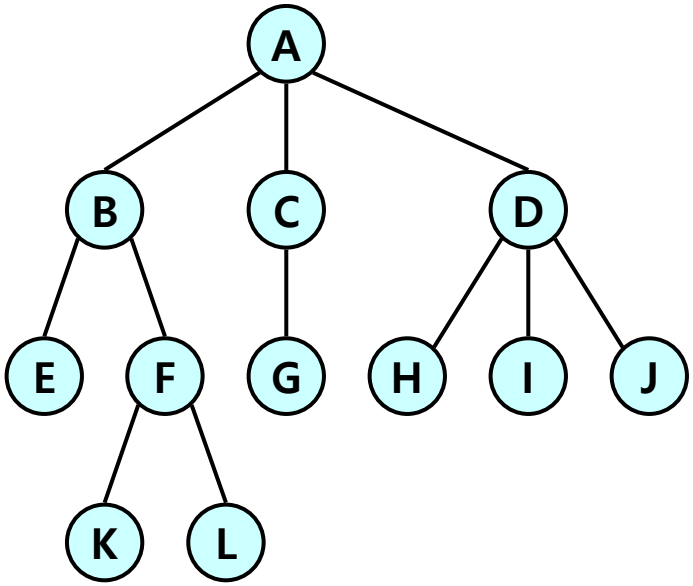
- 트리의 예) 가계도
  - 가계도의 자료: 가족 구성원
  - 자료를 연결하는 선: 부모-자식 관계 표현



- 트리의 정의
  - 트리(tree)는 하나 이상의 노드로 구성된 유한 집합으로서,
  1) 특별히 지정된 노드인 루트(root)가 있고,
  2) 나머지 노드들은 다시 각각이 트리이면서 교차하지 않는 분리집합 $\boldsymbol{T_1, T_2, \ldots, T_m (m \geq 0)}$으로 분할된다.
  - 이 때, $\boldsymbol{T_1, T_2, \ldots, T_m}$을 루트의 서브트리(subtree)라고 한다.

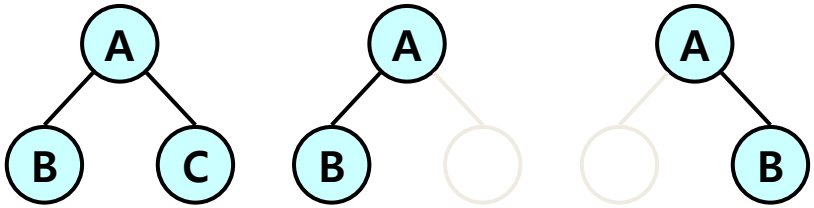


- 노드(node)
  - 트리의 원소
- 간선(edge)
  - 부모 노드와 자식 노드를 연결하는 선
- 루트(root)
  - 트리의 시작 노드
- 단말(leaf, terminal) 노드
  - 자식 노드가 없는 노드
  - 트리 A의 단말 노드 = E, K, L, G, H, I, J



이진 트리(Binary Tree)

- 이진 트리(binary tree)
  - 트리의 노드 구조를 일정하게 정의하여,
    트리의 구현과 연산이 쉽도록 정의한 트리
  - 이진 트리의 모든 노드는
    왼쪽 자식 노드와 오른쪽 자식 노드 만 가질 수 있다.



< 이진 트리의 기본 구조 >

이진 트리의 순회(Traversal)

- 이진 트리의 순회:
  - 트리의 모든 노드를 방문하여 데이터를 처리하는 연산
  - 이진 트리가 순환적으로 정의되어 구성되어있으므로,
    순회작업도 서브트리에 대해서 순환적으로 반복하여 완성한다.
  - 왼쪽 서브트리에 대한 순회를 오른쪽 서브트리 보다 먼저 수행
  - 순회의 종류: 전위 순회, 중위 순회, 후위 순회

전위 순회(Preorder Traversal)

- 전위 순회 수행 방법

  (1) 현재 노드 n을 방문하여 처리

  (2) 현재 노드 n의 왼쪽 서브트리로 이동

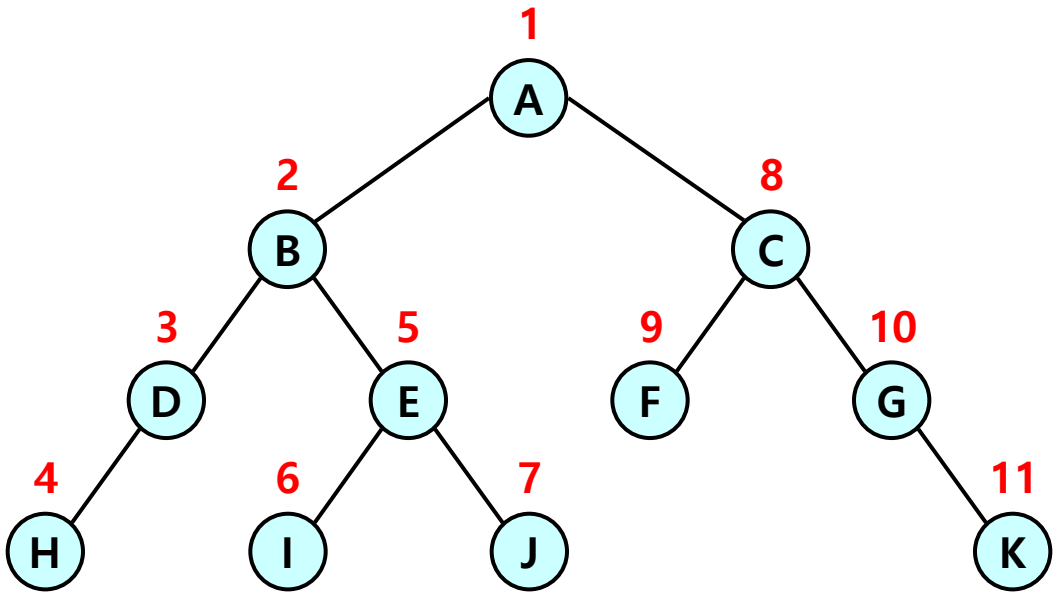
  (3) 현재 노드 n의 오른쪽 서브트리로 이동

- 전위 순회 알고리즘

$$
\begin{aligned}
&\boldsymbol{preorder}(T)\,\{\\
&\quad if\ (T\ =\ null)\ then\ return;\\
&\quad visit\ T.data;\\
&\quad \boldsymbol{preorder}(T.left);\\
&\quad \boldsymbol{preorder}(T.right);\\
&\}
\end{aligned}
$$

전위 순회 예

- 전위 순회 경로



중위 순회(Inorder Traversal)

- 중위 순회 수행 방법

  (1) 현재 노드 n의 왼쪽 서브트리로 이동

  (2) 현재 노드 n을 방문하여 처리

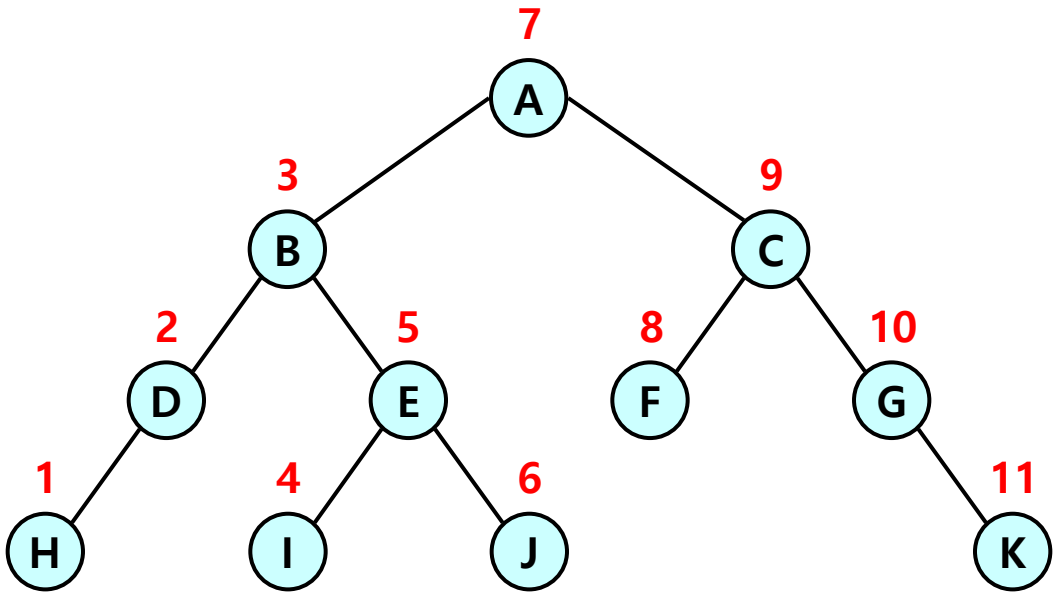
  (3) 현재 노드 n의 오른쪽 서브트리로 이동

- 중위 순회 알고리즘

$$
\begin{aligned}
&\boldsymbol{inorder}(T)\,\{\\
&\quad if\ (T\ =\ null)\ then\ return;\\
&\quad \boldsymbol{inorder}(T.left);\\
&\quad visit\ T.data;\\
&\quad \boldsymbol{inorder}(T.right);\\
&\}
\end{aligned}
$$

중위 순회 예

- 중위 순회 경로



후위 순회(Postorder Traversal)

- 후위 순회 수행 방법

  (1) 현재 노드 n의 왼쪽 서브트리로 이동

  (2) 현재 노드 n의 오른쪽 서브트리로 이동

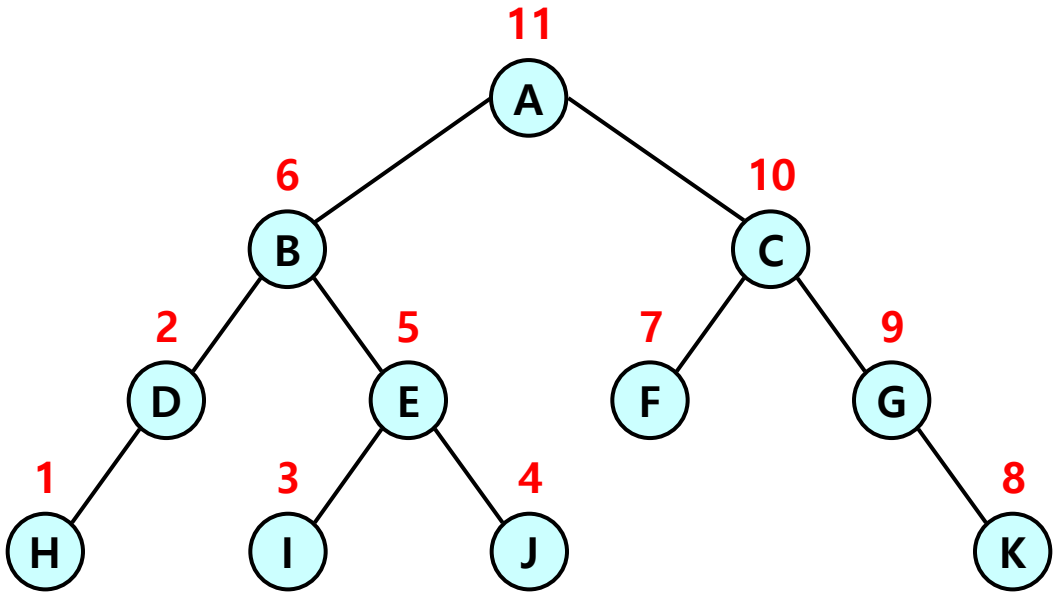
  (3) 현재 노드 n을 방문하여 처리

- 전위 순회 알고리즘

$$
\begin{aligned}
&\boldsymbol{postorder}(T)\,\{\\
&\quad if\ (T\ =\ null)\ then\ return;\\
&\quad \boldsymbol{postorder}(T.left);\\
&\quad \boldsymbol{postorder}(T.right);\\
&\quad visit\ T.data;\\
&\}
\end{aligned}
$$

후위 순회 예

- 후위 순회 경로

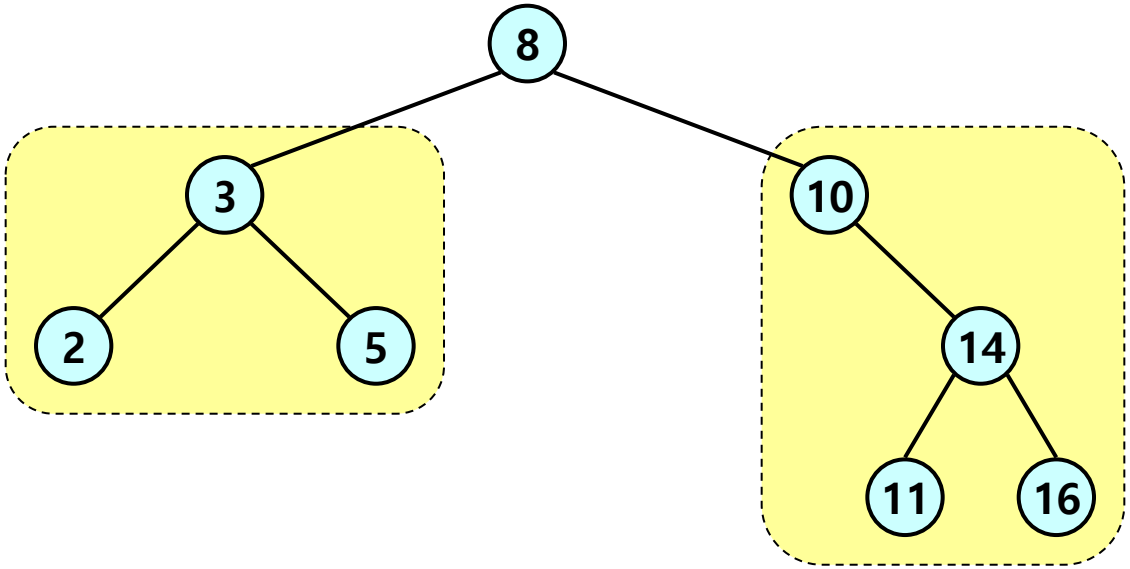


이진 탐색 트리(Binary Search Tree)

- 이진 탐색 트리(binary search tree)
  - 이진 트리에 탐색을 위한 조건을 추가하여 정의한 자료구조

- 이진 탐색 트리의 정의
  - 모든 원소는 서로 다른 유일한 키를 갖는다.
  - 왼쪽 서브트리에 있는 원소의 키들은 그 루트의 키보다 작다.
  - 오른쪽 서브트리에 있는 원소의 키들은 그 루트의 키보다 크다.
  - 왼쪽 서브트리와 오른쪽 서브트리도 이진 탐색 트리이다.

이진 탐색 트리 예

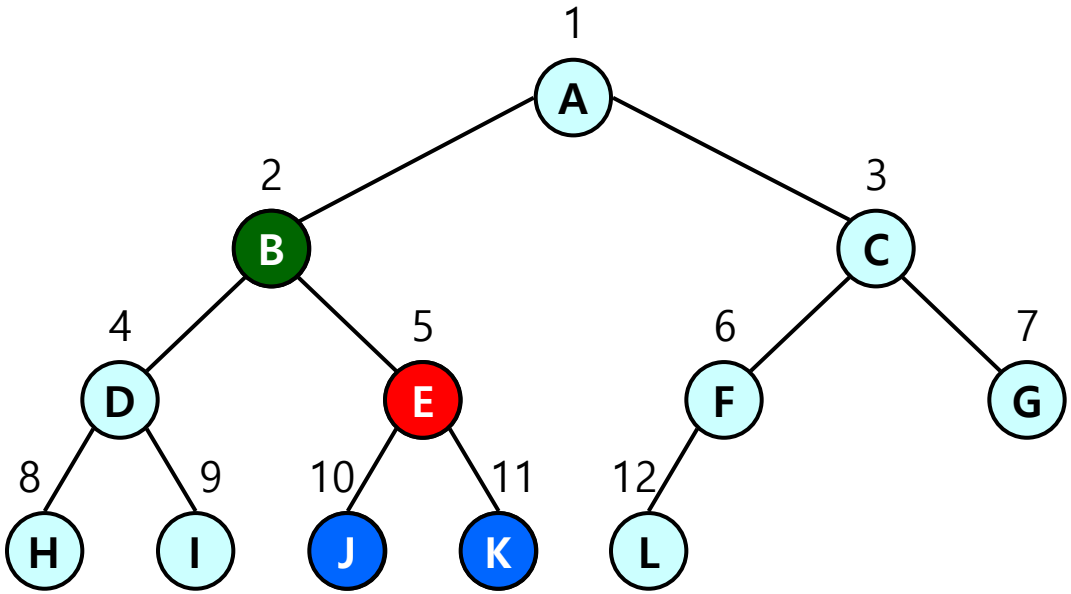


왼쪽 서브트리의 키 값 < 루트의 키 값 < 오른쪽 서브트리의 키 값

트리 구현 1: 순차 자료구조

- 1차원 배열을 이용하여 이진 트리 구현
  - 높이가 h인 포화 이진 트리의 노드 번호를 배열의 인덱스로 사용한다.
  - 인덱스 0번은 사용하지 않고 비워둔다.
  - 인덱스 1번에 루트 저장

- 배열로 구현한 이진 트리



[0]	
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I
[10]	J
[11]	K
[12]	L
[13]	
[14]	
[15]	

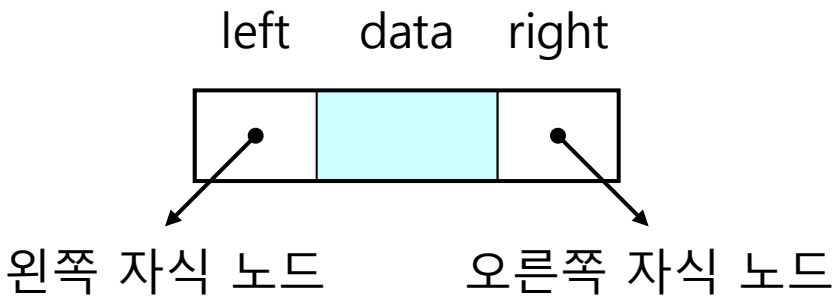
- 이진 트리의 1차원 배열에서의 인덱스 관계 (노드 개수 n)

노드	인덱스	성립 조건
노드 i의 부모 노드		$i > 1$
노드 i의 왼쪽 자식 노드		$2 \times i \leq n$
노드 i의 오른쪽 자식 노드		$(2 \times i) + 1 \leq n$

- 이진 트리의 순차 자료구조 표현의 단점
  - 편향 이진 트리의 경우 메모리 공간 낭비 발생
  - 트리가 계속 커지는 경우 배열의 크기 변경이 어려움

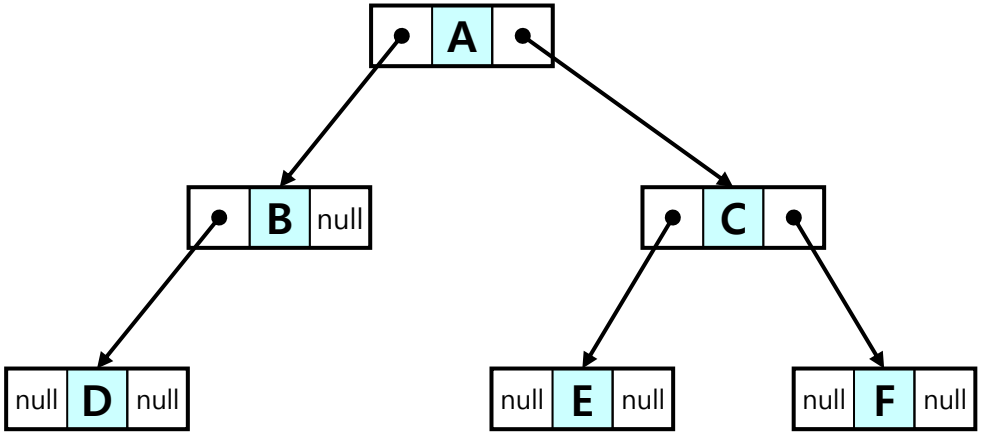
트리 구현 2: 연결 자료구조

- 단순 연결 리스트를 사용하여 이진 트리 구현
  - 이진 트리의 모든 노드는 2개의 자식 노드를 가지므로 일정한 구조의 노드를 사용할 수 있다.




```
typedef struct _Node {
    char data;
    struct _Node *left;
    struct _Node *right;
} Node;
```

- 단순 연결 리스트로 구현한 이진 트리



예제 3.12: 트리 순회

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _Node {
    char data;
    struct _Node *left;
    struct _Node *right;
} Node;

Node *createNode(char data,
            Node *left, Node *right) {
    Node *p = (Node*)malloc(sizeof(Node));
    p->data = data;
    p->left = left;
    p->right = right;
    return p;
}
```

```
void inorder(Node *p) {
    if (p == NULL) return;
    inorder(p->left);
    printf("%c ", p->data);
    inorder(p->right);
}

void main() {
    Node *f = createNode('F', NULL, NULL);
    Node *e = createNode('E', NULL, NULL);
    Node *d = createNode('D', NULL, NULL);
    Node *c = createNode('C', e, f);
    Node *b = createNode('B', d, NULL);
    Node *a = createNode('A', b, c);

    printf("Inorder: ");
    inorder(a);
    printf("\n");
}
```

```
Inorder: D B A E C F
```

Q&A