# 11주차 2차시 이진 탐색 트리 2

## 【학습목표】

1. 이진 트리를 활용하여 탐색을 용이하게 하는 자료구조인 이진 탐색 트리를 설명할 수 있다.
2. 연결 자료구조를 사용하여 프로그래밍할 수 있다.

## 학습내용1 : 연결 자료구조를 이용한 이진 탐색 프로그램 작성 1

### 1. 연결 자료구조를 이용한 이진 탐색의 C 프로그램

```
001  #include<stdio.h>
002  #include<stdlib.h>
003
004  typedef struct treeNode {
005        char key;                  // 데이터 필드
006        struct treeNode* left;     // 왼쪽 서브 트리 링크 필드
007        struct treeNode* right;    // 오른쪽 서브 트리 링크 필드
008  } treeNode;
009
010  typedef char element; // char을 이진 탐색 트리 element의 자료형으로 정의
011
012  treeNode* insertNode(treeNode *p, char x)
013  {  // 포인터 p가 가리키는 노드와 비교하여 노드 x를 삽입하는 연산
014        treeNode *newNode;
015        if (p == NULL){
016           newNode = (treeNode*)malloc(sizeof(treeNode));
017           newNode->key = x;
018           newNode->left = NULL;
019           newNode->right = NULL;
020           return newNode;
021        }
022        else if (x < p->key) p->left = insertNode(p->left, x);
023        else if (x > p->key) p->right = insertNode(p->right, x);
024        else printf("₩n 이미 같은 키가 있습니다! ₩n");
025
026        return p;
027  }
```

* 004~008행 : 트리 노드 구조체 선언

* 012~027행 : 포인터 노드와 비교하여 단말 노드 x를 삽입하는 연산 수행
- 15행 포인터 p가 null이 아니면 현재 비교하고 있는 위치에 노드가 없는 경우
    - 16행 삽입할 노드에 대한 메모리를 할당 받음
    - 17~19행 삽입한 노드의 데이터 필드에 x를 저장하고 삽입하는 노드는 단말 노드이므로 왼쪽 서브 트리와 오른쪽 서브 트리를 NULL 로 설정
    - 20행 삽입 노드를 연결 할 수 있도록 노드의 주소를 반환한다
- 22행 삽입할 데이터 x가 포인터 p가 가리키는 현재 노드의 킷값보다 작은 경우 :
포인터 p가 가리키는 현재 노드의 왼쪽 서브 트리에 대해서 삽입할 자리를 찾기 위해 insertNode() 연산을 재귀호출로 반복한다. 재귀호출을 수행하여 반환된 삽입 노드의 주소(020행)를 현재 노드의 왼쪽 서브 트리, 즉 왼쪽 자식 노드로 설정한다
- 23행 삽입할 데이터 x가 포인터 p가 가리키는 현재 노드이 킷값보다 큰 경우 :
포인터 p가 가리키는 현재 노드의 오른쪽 서브 트리에 대해서 삽입할 자리를 찾기 위해 insertNode() 연산을 재귀호출로 반복한다. 재귀호출을 수행하여 반환된 삽입 노드의 주소(020행)를 현재 노드의 오른쪽 서브 트리, 즉 오른쪽 자식

노드로 설정한다
- 24행 삽입 할 데이터 x가 포인터 p가 가리키는 현재 노드의 킷값과 같은 경우 : 이진 탐색 트리는 같은 킷값의 노드를 가질 수 없으므로 x를 삽입할 수 없다

## 학습내용2 : 연결 자료구조를 이용한 이진 탐색 프로그램 작성 2

### 1. 연결 자료구조를 이용한 이진 탐색의 C 프로그램

```
029  void deleteNode(treeNode *root, element key)
030  { // root 노드부터 탐색하여 key 값과 같은 노드를 찾아 삭제하는 연산
031      treeNode *parent, *p, *succ, *succ_parent;
032      treeNode *child;
033
034      parent=NULL;
035      p=root;
036      while((p != NULL) && (p->key != key)){ // 삭제할 노드의 위치 탐색
037         parent=p;
038         if(key < p->key) p=p->left;
039         else p=p->right;
040      }
041      if(p == NULL){ // 삭제할 노드가 없는 경우
042         printf("₩n 찾는 키가 이진 트리에 없습니다!!");
043         return;
044      }
045
046      // 삭제할 노드가 단말 노드인 경우
047      if((p->left == NULL) && (p->right == NULL)){
048         if(parent != NULL){
049            if(parent->left == p) parent->left=NULL;
050               else parent->right=NULL;
051            }
052         else root=NULL;
053      }
054
```

```
055      // 삭제할 노드가 한 개의 자식 노드를 가진 경우
056      else if((p->left == NULL) || (p->right == NULL)){
057         if(p->left != NULL) child=p->left;
058         else child=p->right;
059
060         if(parent != NULL){
061            if(parent->left == p) parent->left=child;
062             else parent->right=child;
063         }
064         else root=child;
065      }
066
067      // 삭제할 노드가 두 개의 자식 노드를 가진 경우
068      else{
069         succ_parent=p;
070         succ=p->left;
071         while(succ->right != NULL){
072            succ_parent=succ;
073            succ=succ->right;
074         }
075         if(succ_parent->left == succ) succ_parent->left=succ->left;
076         else succ_parent->right=succ->left;
077         p->key=succ->key;
078         p=succ;
079      }
080      free(p);
081  }
082
```

* 029~081행 : 이진 탐색 트리 root 노드부터 탐색하여 key값과 같은 노드를 찾아 삭제하는 연산을 한다
- 036~040행 현재 노드 p와 비교하면서 삭제할 노드의 위치를 탐색하는 작업을 반복
- 041~044행 탐색한 결과 이진 탐색 트리 안에 삭제할 데이터 x가 없는 경우에는 삭제 연산 종료
- 047행 삭제할 노드가 단말 노드 인 경우 :

- 048~051행 삭제할 노드에 부모 노드가 있는 경우에는 부모 노드와 삭제할 노드가 연결된 링크필드를 NULL로 설정
- 52행 삭제할 노드에 부모 노드가 없는 경우에는 삭제하는 노드가 이진 탐색 트리의 하나뿐인 노드이므로 루트 노드를 가리키는 포인터 root를 NULL로 설정

- 56행 삭제할 노드가 한 개의 자식 노드를 가진 경우 :
  - 57행 삭제할 노드가 왼쪽 자식 노드를 가진 경우에는 삭제한 자리를 자식 노드에게 물려주기 위해서 왼쪽 자식 노드에 포인터 child를 설정
  - 58행 삭제할 노드가 오른쪽 자식 노드를 가진 경우에는 삭제한 자리를 자식 노드에게 물려주기 위해서 오른쪽 자식 노드에 포인터 child를 설정
  - 060~064행 삭제할 노드의 부모 노드 parent와 삭제할 노드의 하나뿐인 자식 노드 chile를 연결
- 68행 삭제할 노드가 두 개의 자식 노드를 가진 경우에는 왼쪽 서브 트리에서 후계자 노드를 설정
  - 070행 삭제할 노드의 왼쪽 서브 트리에서
  - 071~074행 오른쪽 링크를 따라가면서 오른쪽 자식 노드가 없는 노드, 즉 가장 큰 노드인 후계자 노드를 찾아 포인터 succ로 지정
  - 075~076행 후계자 노드를 삭제하여 옮기기 전에 후계자 노드의 왼쪽 자식 노드(succ→left)를 후계자 노드의 부모 노드(succ_parent)와 연결
  - 077행 후계자 노드의 데이터 필드 값(succ→key)을 삭제할 노드의 데이터 필드(p→key)로 옮긴다
  - 078행 후계자 노드 succ에 포인터 p를 설정
- 080행 포인터 p의 노드를 메모리 해제하여 후계자 노드를 삭제

## 학습내용3 : 연결 자료구조를 이용한 이진 탐색 프로그램 작성 3

### 1. 연결 자료구조를 이용한 이진 탐색의 C 프로그램

```
083  treeNode* searchBST(treeNode* root, char x)
084  { // 이진 탐색 트리에서 키값이 x인 노드의 위치를 탐색하는 연산
085        treeNode* p;
086        p = root;
087        while (p != NULL){
088           if (x < p->key) p = p->left;
089           else if (x == p->key) return p;
090           else p = p->right;
091        }
092        printf("₩n 찾는 키가 없습니다!");
093        return p;
094   }
095
096  void displayInorder(treeNode* root)
097  { // 이진 탐색 트리를 중위 순회하면서 출력하는 연산
098        if(root){
099           displayInorder(root->left);
100           printf("%c_", root->key);
101           displayInorder(root->right);
102        }
103  }
104
105  void menu()
106  {
107        printf("₩n*-------------------------*");
108        printf("₩n₩t1 : 트리 출력");
109        printf("₩n₩t2 : 문자 삽입");
```

```
110         printf("₩n₩t3 : 문자 삭제");
111         printf("₩n₩t4 : 문자 검색");
112         printf("₩n₩t5 : 종료");
113         printf("₩n*----------------------------*");
114         printf("₩n메뉴입력 >> ");
115     }
116
117     int main()
118     {
119         treeNode* root = NULL;
120         treeNode* foundedNode = NULL;
121         char choice, key;
122
123         root=insertNode(root, 'G'); // 트리 만들기
124         insertNode(root, 'I');
125         insertNode(root, 'H');
126         insertNode(root, 'D');
127         insertNode(root, 'B');
128         insertNode(root, 'M');
129         insertNode(root, 'N');
130         insertNode(root, 'A');
131         insertNode(root, 'J');
132         insertNode(root, 'E');
133         insertNode(root, 'Q');
134
135         while(1){
136           menu();
137           choice = getchar(); getchar();
```

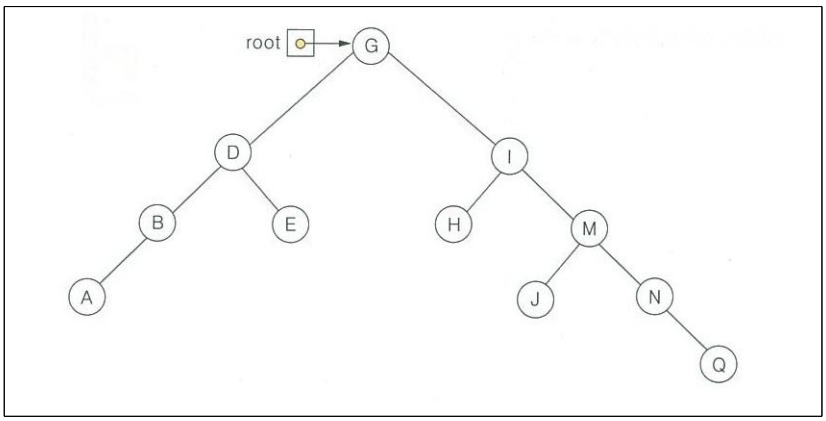
```
139         switch(choice){
140           case 1 : printf("₩t[이진트리 출력] ");
141                    displayInorder(root); printf("₩n");
142                    break;
143
144           case 2 : printf("삽입할 문자를 입력하세요 : ");
145                    key = getchar(); getchar();
146                    insertNode(root, key);
147                    break;
148
149           case 3 : printf("삭제할 문자를 입력하세요 : ");
150                    key = getchar(); getchar();
151                    deleteNode(root, key);
152                    break;
153
154           case 4 : printf("찾을 문자를 입력하세요 : ");
155                    key = getchar(); getchar();
156                    foundedNode = searchBST(root, key);
157                    if (foundedNode != NULL)
158                      printf("₩n %c 를 찾았습니다! ₩n", foundedNode->key);
159                    else printf("₩n 문자를 찾지 못했습니다. ₩n");
160                    break;
161
162           case 5 : return 0;
163           default : printf("없는 메뉴입니다. 메뉴를 다시 선택하세요! ₩n");
164           break;
165           }
166         }
167     }
```

* 083~094행 이진 탐색 트리에서 킷값이 x인 노드의 위치를 탐색하는 연산을 수행
- 087~091행 현재 노드가 NULL이 아닌 동안 현재 노드의 킷값과 x를 비교하여 같으면 현재 노드의 주소 p를 반환
- 093행 x와 같은 노드를 찾지 못하면 NULL로 설정된 포인터 p를 반환

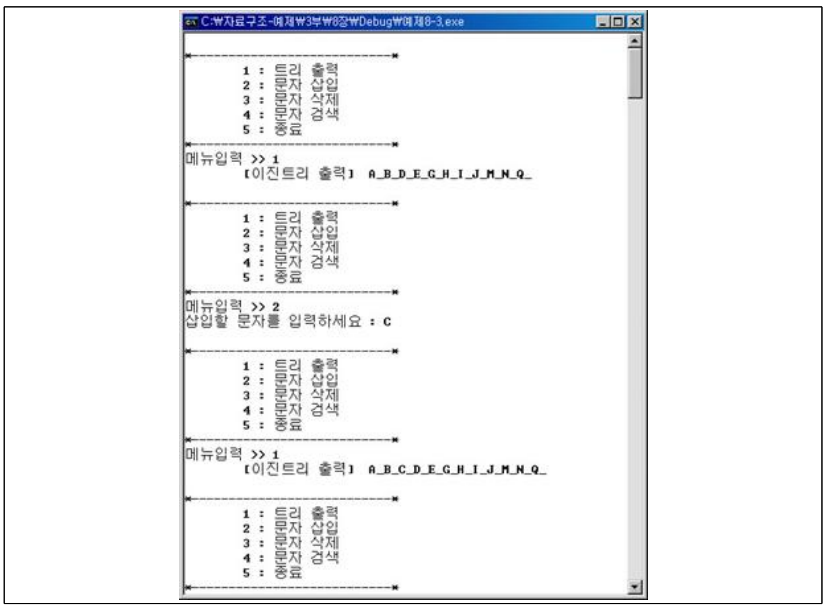
* 096~103행 이진 탐색 트리를 중위 순회하면서 출력하는 연산을 수행

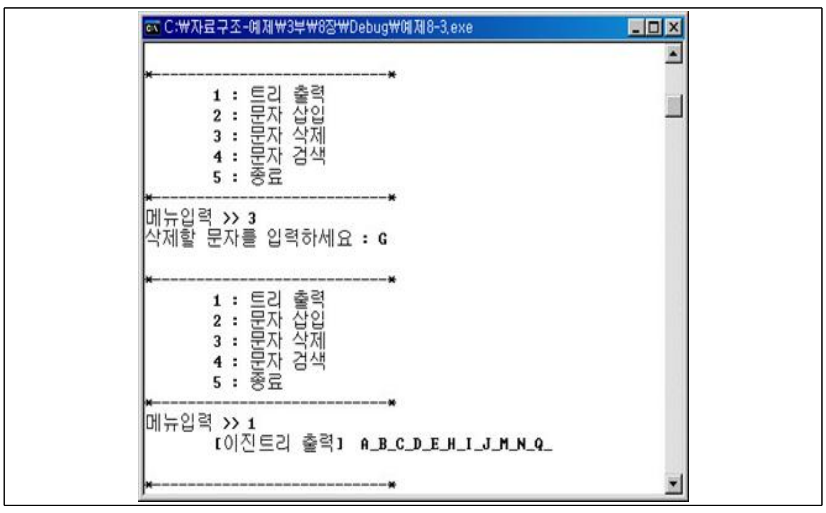
* 123~133행 다음과 같은 초기 이진 탐색 트리를 구성하고 첫 번째로 삽입한 노드 즉 노드 G를 루트 노드 포인터 root로 지정

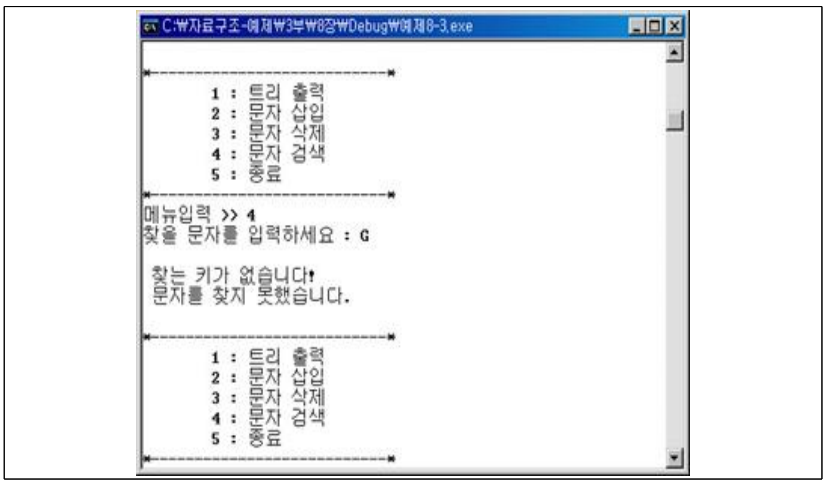
* 139~165행 메뉴 선택 연산


root
G
D
I
B
E
H
M
A
J
N
Q

* 실행 결과


C:₩자료구조-예제₩3부₩8장₩Debug₩예제8-3.exe
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
메뉴입력 >> 1
[이진트리 출력] A B D E G H I J M N Q
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
메뉴입력 >> 2
삽입할 문자를 입력하세요 : C
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
메뉴입력 >> 1
[이진트리 출력] A B C D E G H I J M N Q
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료


C:₩자료구조-예제₩3부₩8장₩Debug₩예제8-3.exe
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
메뉴입력 >> 3
삭제할 문자를 입력하세요 : G
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
메뉴입력 >> 1
[이진트리 출력] A B C D E H I J M N Q




**【학습정리】**

1. 이진 트리를 활용하여 탐색을 용이하게 하는 자료구조인 이진 탐색 트리를 정의하고 연결 자료구조를 사용하여 프로그래밍해 보았다.