〈 소스코드 〉

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

class BstNode;  // 전방선언
class BST {
private:
    BstNode *root;
public:
    BST() { root = 0; } // 생성자 함수
    BstNode *IterSearch(const int&);        // 탐색 함수
    void inorder();                         // 중위우선순회 함수
    void inorder(BstNode *CurrentNode);     // 중위우선순회 함수
    bool Insert(const int&);                // 삽입 함수
    bool Delete(const int&);                // 삭제 함수
};

class BstNode {
    friend class BST;                       // friend 선언
private:
    BstNode *LeftChild;
    int data;
    BstNode *RightChild;
public:
    BstNode(int element = 0, BstNode *left = 0, BstNode *right = 0) {
        data = element;         // 변수 초기화
        LeftChild = left;
        RightChild = right;
    }
};

// 탐색 함수
BstNode* BST::IterSearch(const int&x) {
    for (BstNode *t = root; t;) {   // t가 rrot부터 탐색 시작
        if (x == t->data) return t;
        if (x < t->data) t = t->LeftChild;
        else t = t->RightChild;
    }
    return 0;
};

void BST::inorder() {   // 중위우선쉬회 함수
    inorder(root);
};

void BST::inorder(BstNode* CurrentNode)     // 중위우선순회 함수
{
    if (CurrentNode) {
        inorder(CurrentNode->LeftChild);    // 왼쪽 자식으로 이동
        cout << CurrentNode->data << " ";   // data 출력
        inorder(CurrentNode->RightChild);   // 오른쪽 자식으로 이동
    }
};

// 삽입 함수
bool BST::Insert(const int &x) {
    BstNode *p = root;
    BstNode *q = 0; // p를 뒤따라오는 노드

    while (p) { // 탐색
        q = p;
        if (x == p->data) { // 삽입하려는 x가 이미 존재하는 경우
            cout << x << "가 이미 존재합니다." << endl;
            return false;
        }
        if (x < p->data)
            p = p->LeftChild;
        else p = p->RightChild;
    }
    p = new BstNode;    // 삽입
    p->LeftChild = p->RightChild = 0;   // p의 왼쪽, 오른쪽 자식 NULL 저장
    p->data = x;

    if (!root) root = p;    // 빈 리스트
```

```
74          else if (x < q->data)
75              q->LeftChild = p;
76          else q->RightChild = p;
77          return true;    // 삽입 성공 시 true
78      };
79
80      // 삭제 함수
81      bool BST::Delete(const int&x) {
82          BstNode *p = root;
83          BstNode *q = 0; // p를 뒤따라오는 노드
84
85          while (p) { // 빈 리스트가 아닌 경우
86              if (x < p->data) {  // x가 p의 data보다 작은 경우
87                  q = p;
88                  p = p->LeftChild;   // 왼쪽 자식으로 이동
89              }
90              else if (x > p->data) { // x가 p의 data보다 큰 경우
91                  q = p;
92                  p = p->RightChild;  // 오른쪽 자식으로 이동
93              }
94              else    // x가 p의 data와 같은 경우
95                  break;
96          }
97
98          if (!p) {   // x와 일치하는 값이 없는 경우
99              cout << "삭제할 노드가 없습니다." << endl;
100             return false;
101         }
102
103         if (p->LeftChild == 0 && p->RightChild == 0) {  // 삭제할 노드의 자식이 없는 경우(단말노드)
104             if (x < q->data) {  // x가 q의 data보다 작은 경우
105                 q->LeftChild = 0;   // q의 왼쪽 자식 NULL
106                 delete p;   // p 삭제
107             }
108             else {
109                 q->RightChild = 0;
110                 delete p;
111             }
112         }
113         else if (p->LeftChild == 0 || p->RightChild == 0) { // 삭제할 노드의 자식이 하나인 경우
114             if (p->LeftChild == 0) {    // 오른쪽 자식이 있는 경우
115                 if (x < q->data) {  // p가 q의 왼쪽 자식인 경우
116                     q->LeftChild = p->RightChild;   // p의 오른쪽 자식을 q의 왼쪽 자식으로
117                     delete p;   // p 삭제
118                 }
119                 else {  // p가 q의 오른쪽 자식인 경우
120                     q->RightChild = p->RightChild;  // p의 오른쪽 자식을 q의 오른쪽 자식으로
121                     delete p;   // p 삭제
122                 }
123             }
124             else {  // 왼쪽 자식이 있는 경우
125                 if (x < q->data) {  // p가 q의 왼쪽 자식인 경우
126                     q->LeftChild = p->LeftChild;    // p의 왼쪽 자식을 q의 왼쪽 자식으로
127                     delete p;   // p 삭제
128                 }
129                 else {  // p가 q의 오른쪽 자식인 경우
130                     q->RightChild = p->LeftChild;   // p의 왼쪽 자식을 q의 오른쪽 자식으로
131                     delete p;   // p 삭제
132                 }
133             }
134         }
135         else {  // 삭제할 노드의 자식이 둘인 경우
136             BstNode *r = p; // p를 r에 저장
137             q = p;  // q는 p를 뒤따라오는 노드
138             p = p->RightChild;  // p는 오른쪽 자식으로 이동
139
140             while (p->LeftChild) {
141                 q = p;
142                 p = p->LeftChild;
143             }
144             r->data = p->data;  // p의 data를 r에 저장
145
```

```
146         if (r == q) {
147             r->RightChild = p->RightChild;  // p의 오른쪽 자식을 r의 오른쪽 자식으로
148             delete p;   // p 삭제
149         }
150         else if (p->RightChild) {   // p의 오른쪽 자식이 있는 경우
151             q->LeftChild = p->RightChild;   // p의 오른쪽 자식을 q의 왼쪽 자식으로
152             delete p;   // p 삭제
153         }
154         else {  // p가 단말노드인 경우
155             q->LeftChild = 0;   // q의 왼쪽자식 NULL
156             delete p;   // p 삭제
157         }
158     }
159     return true;
160 };
161
162 int main() {
163     BST bst;
164     int select, x, val;
165
166     cout << "<<<<<<<<<<<<<<<<<<< 이진탐색 트리 >>>>>>>>>>>>>>>>>>>" << endl;
167     while (1) {
168         cout << "(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료" << endl;
169         cin >> select;
170
171         switch (select) {
172             case 1: // 삽입 함수 호출
173                 cout << "입력할 키 개수 : ";
174                 cin >> x;
175                 cout << "삽입할 키를 입력 : ";
176                 for (int i = 0; i < x; i++) {
177                     cin >> val;
178                     bst.Insert(val);
179                 }
180                 break;
181             case 2: // 삭제 함수 호출
182                 cout << "삭제할 값 입력 : ";
183                 cin >> val;
184                 bst.Delete(val);
185                 break;
186             case 3: // 탐색 함수 호출
187                 cout << "탐색할 값 입력 : ";
188                 cin >> val;
189                 if (bst.IterSearch(val) == 0)
190                     cout << val << " ==> 탐색 실패" << endl;
191                 else
192                     cout << val << " ==> 탐색 성공" << endl;
193                 break;
194             case 4: // 중위우선순회 함수 호출
195                 cout << "중위우선순회 : ";
196                 bst.inorder();
197                 cout << endl;
198                 break;
199             case 5: // 5 입력 시 종료
200                 return 0;
201         }
202     }
203 }
```

〈 실행화면 〉

```
C:₩Users₩LEE KUNUK₩Desktop₩2-2의 건욱이₩고급프로그래밍₩2-2 C₩Debug₩2-2 C.exe
<<<<<<<<<<<<<<<<<<<< 이진탐색 트리 >>>>>>>>>>>>>>>>>>>>
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
1
입력할 키 개수 : 10
삽입할 키를 입력 : 48 3 241 55 2 6 88 39 21 100
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
4
중위우선순회 : 2 3 6 21 39 48 55 88 100 241
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
3
탐색할 값 입력 : 100
100 ==> 탐색 성공
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
3
탐색할 값 입력 : 101
101 ==> 탐색 실패
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
2
삭제할 값 입력 : 88
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
4
중위우선순회 : 2 3 6 21 39 48 55 100 241
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
2
삭제할 값 입력 : 1
삭제할 노드가 없습니다.
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
4
중위우선순회 : 2 3 6 21 39 48 55 100 241
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
1
입력할 키 개수 : 1
삽입할 키를 입력 : 200
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
4
중위우선순회 : 2 3 6 21 39 48 55 100 200 241
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
2
삭제할 값 입력 : 2
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
4
중위우선순회 : 3 6 21 39 48 55 100 200 241
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
3
탐색할 값 입력 : 200
200 ==> 탐색 성공
(1) 삽입  (2) 삭제  (3) 탐색  (4) 중위우선순회  (5) 종료
```