B-tree

- **B-tree**의 삽입 알고리즘 **insertBT(T, m, newKey)**과 삭제 알고리즘 **deleteBST(T, m, oldKey)**을 구현하시오.
  - 단 알고리즘은 수업 슬라이드에서 설명한 로직을 이용해 구현하시오.

- 출력을 위해 **B-tree**의 **inorder** 순회 알고리즘을 구현하시오
  - inorderBT(T, m)

- 다음 페이지에서 주어진 삽입 및 삭제 시퀀스에 대해, **m=3**일 때와 **m=4**일 때에 대해 각각 실행한 결과를 **inorderBT** 함수를 이용하여 출력하시오.

◆ 노드 삽입 순서
- 30, 20, 62, 110, 140, 15, 65, 136, 150, 120,
- 40, 132, 19, 128, 138, 100, 16, 145, 70, 42,
- 69, 43, 26, 60, 130, 50, 18, 7, 36, 58,
- 22, 41, 59, 57, 54, 33, 75, 124, 122, 123

◆ 노드 삭제 순서
- 40, 132, 19, 128, 138, 100, 16, 145, 70, 42,
- 22, 41, 62, 110 140 15 65 124, 122, 123,
- 30, 20, 59, 57, 54, 33, 75, 136, 150, 120,
- 69, 43, 26, 60, 130, 50, 18, 7, 36, 58

◆ 출력
- 삽입과 삭제 모두 노드 하나를 삽입 삭제할 때마다 트리의 inorder 순회 순서를 출력하시오.
- 따라서 다음과 같은 main 루틴을 이용하시오.

- **Main (Report 2, 삽입)**
  - // m=3일 때 삽입
  - T = null;
  - insertBT(T, 3, 30); inorderBT(T, 3);
  - insertBT(T, 3, 20); inorderBT(T, 3);
  - insertBT(T, 3, 62); inorderBT(T, 3);
  - . . . .
  - insertBT(T, 3, 123); inorderBT(T, 3);

  - // m=4일 때 삽입
  - T = null;
  - insertBT(T, 4, 30); inorderBT(T, 4);
  - insertBT(T, 4, 20); inorderBT(T, 4);
  - insertBT(T, 4, 62); inorderBT(T, 4);
  - . . . .
  - insertBT(T, 4, 123); inorderBT(T, 4);

- **Main (Report 3, 삭제)**
  - // m=3일 때 트리 생성
  - T = null;
  - insertBT(T, 3, 30);
  - insertBT(T, 3, 20);
  - ertBT(T, 3, 62);
  - . . . .
  - insertBT(T, 3, 123);

  - // m=3일 때 트리 삭제
  - deleteBT(T, 3, 40); inorderBT(T, 3);
  - deleteBT(T, 3, 132); inorderBT(T, 3);
  - deleteBT(T, 3, 19); inorderBT(T, 3);
  - . . . .
  - deleteBT(T, 3, 58); inorderBT(T, 3);

- // m=4일 때 트리 생성
- T = null;
- insertBT(T, 4, 30);
- insertBT(T, 4, 20);
- insertBT(T, 4, 62);
- . . . .
- insertBT(T, 4, 123);

- // m=4일 때 트리 삭제
- deleteBT(T, 4, 40); inorderBT(T, 4);
- deleteBT(T, 4, 132); inorderBT(T, 4);
- deleteBT(T, 4, 19); inorderBT(T, 4);
- . . . .
- deleteBT(T, 4, 58); inorderBT(T, 4);

- 제출물
  - 표지
  - 출력 결과 (화면 덤프)
  - 프로그램 소스