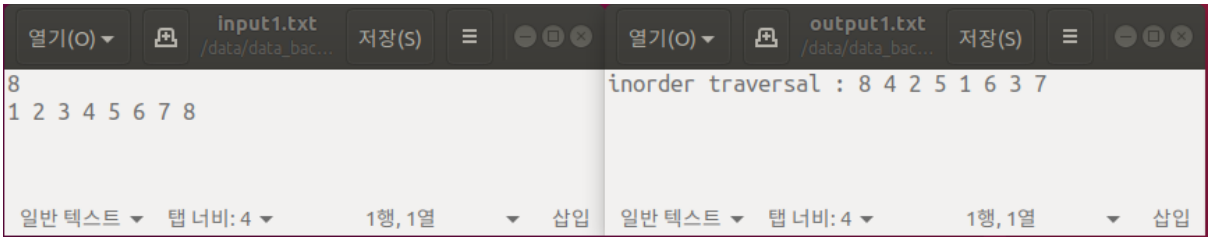
2023 Data Structure (CSE2010 12282)

Lab 04 (3/31) Threaded Tree & Traversal

- arranged by TA Beomsoo Kim

Due : ~2023.04.05(Wed) 23:59, Late Submission : ~2023.04.06.(Thu) 23:59

- **Threaded Tree & Inorder Traversal Implementation**
- 아래의 사진과 같이 **[input].txt** 파일을 입력받아 **Threaded Tree**를 생성하고 편집하는 기능을 구현하고**, Inorder Traversal**의 결과 값을 **[output].txt** 파일에 출력하여 저장한다**.**
- **Implement the function of creating and editing a Threaded Tree by receiving the [input].txt file as input as below and the output result of Inorder Traversal is stored in the [output].txt file.**



<input>

- 첫째줄 : Threaded Tree에 insert할 Node의 개수 (100 이하의 자연수)
- 둘째줄 : Threaded Tree에 삽입할 Node들의 Key값 (10000 이하의 자연수, 공백 한칸씩을 사이에 두고 작성됨)
- first line : the number of nodes to be inserted into Threaded Tree (Natural Number less or equal to 100)
- second line : the key values of the nodes to be inserted into Threaded Tree (All the values are typed with single space between each other in a line, and the key value is a natural number less or equal to 10000)

<output>

- 생성된 **Threaded Tree**를 **Inorder**로 **Traverse**한 결과 출력**.**
- **“inorder traversal : ”** 이후 숫자 출력**.** 출력 포맷이 틀리면 해당 테스트 케이스에 대해서 **0**점**.**
- 숫자 사이에 한칸의 간격을 둘것 **->** 역시 포맷이 틀리면 **0**점
- **Inorder Traversal result of generated Threaded Tree.**
- **Print number after printing “inorder traversal: ”. If the output format is wrong, 0 points for that test case.**
- **Make sure to leave a space between the numbers.**

<CreateTree>

- 새로운 Threaded Tree를 만드는 함수입니다.
- 메모리를 할당한 Threaded Tree포인터를 반환합니다.
- **left_thread :** left_child 포인터가 thread인지에 대한 flag입니다. thread이면 1, thread가 아니라 자식을 가리키면 0입니다.
- **right_thread :** right_child 포인터가 thread인지에 대한 flag입니다. thread이면 1, thread가 아니라 자식을 가리키면 0입니다.
- **left_child :** threads나 왼쪽 자식 노드를 가리킬 수 있는 포인터 입니다.
- **right_child :** threads나 오른쪽 자식 노드를 가리킬 수 있는 포인터 입니다.
- **data :** Thread Tree 노드의 데이터 값입니다. root node의 data는 -1로 초기화 합니다.
- This function creates a new Threaded Tree

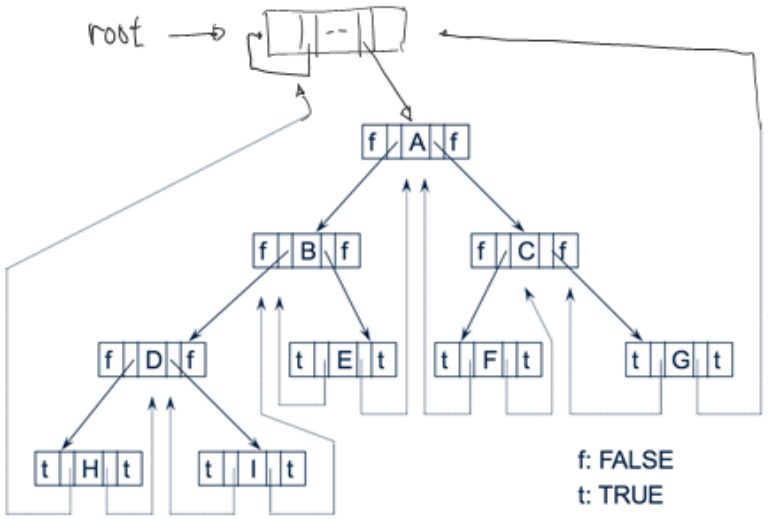
- Returns a pointer to the Threaded Tree that allocated the memory
- left_thread : A flag indicating if the left_child pointer is a thread. 1, if it is a thread or 0, if it points to a child node.
- right_thread : A flag indicating if the right_child pointer is a thread. 1, if it is a thread or 0, if it points to a child node.
- left_child : Pointer to threads or left child node
- right_child : Pointer to threads or right child node
- data : The data value of the Threaded Tree node. The data of the root node is initialized to -1.

<insert>

- 입력한 Node들의 key값을 Threaded Tree에 삽입하는 함수입니다.
- Threaded Tree에 값들을 Insert 할때는 입력파일의 왼쪽 숫자 부터 index 순서대로 입력해주셔야 합니다.

  ex) **input : "A B C D E F G H I"** 일때 아래의 사진과 같이 **Complete Binary Tree**의 형태로 Insert 하시면 됩니다.
- **root :** Threaded Tree의 루트노드를 가리키는 포인터
- **root_idx :** Threaded Tree의 root노드의 index, 삽입할 곳을 찾는 최초의 노드를 의미합니다.
- **data :** 새로 삽입할 노드의 데이터 값
- **idx :** 새로 삽입할 노드의 인덱스 값, 0은 root index를 의미
- 삽입에 성공했으면 1을 반환하고, 실패하면 0을 반환합니다.
- 삽입에 실패할 경우 main함수에서 "Insertion failed!" error message가 출력되어야 합니다.
- This function inserts the key values of the entered nodes into the Threaded Tree.
- When inserting key values into the Tree, it should be inserted in the order of index from the left to right.

  ex) **input : "A B C D E F G H I"** -> Insertion should be done as **Complete Binary Tree** shape as below.
- **root :** Pointer to the root node of Threaded Tree
- **root_idx :** index of the root node of Threaded Tree, means the first node to find the insertion position.
- **data :** data key value of the newly inserted node
- **idx :** The index of the node to be newly inserted, 0 means root index.
- Return 1 if the insert was successful, 0 if unsuccessful.
- If insertion fails, "Insertion failed!" An error message should be printed, in main function



- 위 사진의 경우**, root**의 **index**는 **0**, **A**의 **index** 는 **1**, **B**의 **index**는 **2**, **C**의 **index**는 **3**, **D**의 **index**는 **4**, **E**의 **index**는 **5**, **F**의 **index**는 **6**, **G**의 **index**는 **7**, **H**의 **index**는 **8**, **I**의 **index**는 **9** 입니다.
- In the threaded tree example above, index of root is 0, index of A is 1, index of B is 2, index of C is 3, index of D is 4, index of E is 5, index of F is 6, index of G is 7, index of H is 8, and the index of I is 9.

<DeleteTree>

- Threaded tree의 root를 입력으로 받아 해당 트리를 삭제하는 함수입니다.
- This function receives the root of the threaded tree as input and deletes the tree.

<printInorder>

- **Inorder**순서로 출력해주시면 됩니다**.** 출력시 각 **key**당 간격은 한칸입니다**.**
- **printInorder()** 함수의 경우**, Recursive** 형태의 구현은 금지됩니다**. (ex: F()** 함수 내부에서 똑같은 **F()**함수를 호출하는 행위 금지**.) - Recursive** 형태 사용시 **0**점처리 됩니다**.**
- **Print out inorder. When printing out, the interval between each key is one space.**
- **In the case of the function "printInorder()", Implementing in a recursive form is prohibited. (ex: calling F() inside of F() is banned.) - 0 score for Recursive form.**

<Structure & Function Format>

Structure

```
struct ThreadedTree {
    int left_thread; // flag if ptr is thread
    ThreadedPtr left_child;
    ElementType data;
    ThreadedPtr right_child;
    int right_thread; // flag if ptr is thread
}ThreadedTree;
```

Function

```
ThreadedPtr CreateTree();
int Insert(ThreadedPtr root, int root_idx,
        ElementType data, int idx);
void printInorder(ThreadedPtr root);
void DeleteTree(ThreadedPtr root);
```

- 위 사진과 같은 Struct 구조체를 사용하셔야 합니다.
- 위 사진과 같은 함수들을 형식에 맞게 구현해주시면 됩니다.
- Struct format above should be used for implementation
- Functions should be implemented in appropriate format as above.

<File Name Format>

- [StudentID].c   ex) 20XXXXXXXX.c

<Execution>

- gcc 20XXXXXXXX.c -o 20XXXXXXXX
- ./20XXXXXXXX [input_file_name] [output_file_name]
- !!! 꼭 제공되는 testcase로 실행시켜보시기 바랍니다.!!!!!,
- !!! Run your solution code with the provided test case above and check whether it works properly !!!
-

<Issue>

- 코드 작성시 주석을 적어주시기 바랍니다. 주석이 없는경우 Cheating으로 간주될 수 있습니다.
- 제공된 testcase는 채점 case에 포함됩니다. 모두 알맞게 나오는지 확인해보시기 바랍니다.
- 파일 입출력은 *argv[]* 를 사용하여 구현해주시기 바랍니다.
- 제출 마감 시간 이전의 가장 최신 버전의 commit을 기준으로 채점할 예정입니다.
- 제출 파일과, 폴더 naming 은 꼭 지정된 형식으로 해주셔야 합니다.
- Please write down the detailed comments when writing the code. If there is no comment, it might be considered cheating.
- Provided test case is included in the test case for grading. Please check to see if it makes a proper result.
- Do not use a fixed file name when inputting and outputting files, but implement it using argv[] as in skeleton code.
- Scoring will be based on the latest version of commit before the deadline.
- The names of the .c file and directory should be named in proper format.

- 출력시, 위 사진의 예시와 같은 형식으로 출력해주시면 됩니다. 모든 공백은 띄어쓰기 한칸입니다. 모든 출력 메시지의 알파벳은 소문자만 사용하여 출력 합니다.
- ->출력파일 첫째줄의 마지막 숫자 출력후 EOF.
- All the messages must be printed out according to the appropriate format as shown in the example above. All spaces are one space. Only lowercase letters should be used for the alphabets in output messages.
- ->EOF right after the last key value in the line in the output file.

<Directory Format>

- 아래와 같이 git 프로젝트 폴더에 “lab04” 폴더 생성후, "lab04" 폴더 안에 "20XXXXXXXX.c" 파일을 위치시키시면 됩니다.
- After creating the "lab05" directory in the git-project-directory as below, place the "20XXXXXXXXXX.c" file in the "lab04" directory.

>>

2023_CSE2010_20XXXXXXXX/ (GitLab project directory)
---...
---lab03/
-------2023XXXXXX.c
——---...
---lab04/
-------2023XXXXXX.c