실습 과제 #4: Threaded Binary Tree

2022. 1학기

※ 제출 관련 사항

(1) 제출기한: 2022. 5. 25(수), 오후 5시 (늦은 제출은 받지 않음)

(2) 제출파일

학번.c (또는 학번.cpp)

학번.pdf (보고서: 코드 실행 결과 캡쳐 + 자신의 말로 코드/알고리즘/시간복잡도 설명한 것. 두 가지 내용이 모두 있어야 한다. 분량 제한 A4 2장)

(3) 채점환경: 비주얼스튜디오2019 (기타 환경에서 코드 작성하였어도 비주얼스튜디오에서 컴파일 가능한지 반드시 확인할 것)

※과제 상세설명

5.5절 연습문제 #2, insertLeft 구현한다.

1) **함수 initTree**: 입력파일로부터 수행할 동작을 읽어 들여, threaded binary tree를 구성하고 이를 return하는 함수이다. 입력파일의 양식은 아래 **(1)입력**과 같다. 입력파일을 읽기 전 head node가 되는 root부터 생성해야 한다. root의 초기 leftChild 및 rightChild는 root이며, 즉 스스로를 가리킨다. 또한 root의 초기 leftThread와 rightThread는 각각 true, false이다. 입력파일을 읽어 들이면서 node를 추가하여 최종적으로 head node인 root를 return한다.
2) **함수 inpredec**: inorder traversal를 수행할 때, traversal 순서상 주어진 node 바로 앞에 방문하는 node를 찾아 그 포인터를 return하는 함수이다. 즉 함수 insucc의 반대 역할을 수행하는 것로, 교재에서 insucc 함수를 참고하여 작성한다.
3) **함수 insertLeft**: threaded binary tree의 한 node의 왼쪽 자식으로 새로운 node를 삽입하는 함수이다. insertRight 함수의 동작을 응용하여 작성한다.
4) **함수 tinorderSearch**: threaded binary tree의 노드 중에서 탐색하고자 하는 값과 일치하는 data를 가진 node의 포인터를 return한다. 탐색 시에는 반드시 inorder-traversal 방식을 이용해야 하며, 교재에 구현된 tinorder를 응용하여 작성한다.
5) **함수 tpreorder**: threaded binary tree의 구조를 사용해, threaded binary tree를 preorder traverse 하는 함수를 작성한다. traversal을 수행하면서 node의 data를 출력하도록 한다.

(1)입력: txt 파일로 주어지고, threaded binary tree를 구성하기 위한 동작들이 명시되어 있다. 동작들은 아래와 같다. 함수 initTree에서 입력 받아 적절히 동작을 수행하여 threaded binary tree를 구성한다.

- 첫 번째 column:
  - 'I'일 때, insert operation을 수행한다.
  - 'S'일 때, search operation을 수행한다.
- 두 번째 column:
  - Insert하거나 search할 data가 명시된다.
- 셋 번째 column: 첫 번째 column이 'I'인 경우에만 명시된다.
  - 'L': search operation을 통해 찾은 node의 왼쪽에 새 노드를 삽입한다.

▪ 'R': search operation을 통해 찾은 node의 오른쪽에 새 노드를 삽입한다.

▶ 입력파일의 예 (input.txt)

```
I A L
S A
I B L
I C R
S B
I D L
I E R
S C
I F R
S E
I G L
```

채점 시에는 위와 다른 txt 파일을 이용할 수 있다.

(2) 자료구조와 알고리즘 (아래 코드 복사하여 사용하고, 주의사항 반드시 확인할 것)

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

 /* 아래 구조체 제외하고 전역변수 사용 금지
 전역변수 사용 시 0점 처리 */
typedef struct threadedTree* threadedPointer;
typedef struct threadedTree {
        char data;
        short int leftThread, rightThread;
        threadedPointer leftChild, rightChild;
} threadedTree;

 /* 구현할 함수들
 함수의 반환형이나 parameter는 수정 가능하지만, 함수명은 변경 불가
 함수명 변경 시 해당 함수 미구현으로 간주하고 점수 미부여 */
threadedPointer initTree(FILE* fp); // 직접 구현
threadedPointer insucc(threadedPointer tree); // 교재 확인
threadedPointer inpredec(threadedPointer tree); // 직접 구현
void insertRight(threadedPointer parent, threadedPointer child, char data); // 교재 확인
void insertLeft(threadedPointer parent, threadedPointer child, char data); // 직접 구현
threadedPointer tinorderSearch(threadedPointer tree, char data); // 직접 구현
void tinorder(threadedPointer tree); // 교재 확인
```

```
void tpreorder(threadedPointer tree); // 직접 구현

void main() {
        /* main 함수는 수정할 수 없음
        main 함수 수정 시 0점 처리 */
        FILE* fp = fopen("input.txt", "r");
        threadedPointer my_tree = initTree(fp);

        tinorder(my_tree);
        printf("₩n");
        tpreorder(my_tree->leftChild);
        printf("₩n");

        fclose(fp);
        return;
}
```

(3) 출력: (stdout 가정)

```
D B G E A C F
A B D E G C F
```

(참고) 입출력 예시 추가

input1.txt

```
I A L
S A
I B L
I C R
S B
I D L
I E R
S C
I F L
I G R
S D
I H L
I I R
```

output1

```
H D I B E A F C G
A B D H I E C F G
```

input2.txt

```
I A L
S A
I B L
I K R
S K
I C R
S B
I D L
I E R
S C
I F L
I G R
S D
I H L
I I R
```

output2

```
H D I B E A K F C G
A B D H I E K C F G
```

input3.txt

```
I A L
S A
I B L
I C R
S B
I K L
I E R
S C
I F L
I G R
S K
I D L
S D
I H L
I I R
```

output3

```
H D I K B E A F C G
A B K D H I E C F G
```