Homework 10

https://github.com/YuseongJin/homework10

2018038022

소프트웨어학과

진유성

.vscode > C bst-2-full.c

```c
/*Binary Search Tree #2*/

#include <stdio.h>//standard input/output library, 표준 입출력 라이브러리
#include <stdlib.h>//c표준 유틸리티 함수들을 모아 놓은 헤더파일

typedef struct node {//구조체 선언
    int key;//int 형 변수 key 멤버 선언
    struct node *left;//자기참조 구조체, left
    struct node *right;//자기참조 구조체, right
} Node;//구조체 별칭

/* for stack */
#define MAX_STACK_SIZE      20
//MAX_STACK_SIZE를 20 으로 정의
Node* stack[MAX_STACK_SIZE];
int top = -1;
//int형 변수 top선언후 -1삽입

Node* pop();
void push(Node* aNode);

/* for queue */
#define MAX_QUEUE_SIZE      20
//MAX_QUEUE_SIZE를 20 으로 정의
Node* queue[MAX_QUEUE_SIZE];
int front = -1;
//int형 변수 front선언후 -1삽입
int rear = -1;
//int형 변수 rear선언후 -1삽입

Node* deQueue();
void enQueue(Node* aNode);


int initializeBST(Node** h);

/* functions that you have to implement */
void recursiveInorder(Node* ptr);     /* recursive inorder traversal */
void iterativeInorder(Node* ptr);     /* iterative inorder traversal */
void levelOrder(Node* ptr);           /* level order traversal */
int insert(Node* head, int key);      /* insert a node to the tree */
int deleteNode(Node* head, int key);  /* delete the node for the key */
int freeBST(Node* head); /* free all memories allocated to the tree */

/* you may add your own defined functions if necessary */


void printStack();

```

```c
//함수 원형선언

int main()//메인함수
{
    char command;//char형 변수 선언, 명령어
    int key;//int형 변수 선언, 키보드
    Node* head = NULL;//node head의 값을 선언후 NULL로 초기화

    printf("[----- [Yuseong Jin] [2018038022] -----]");//이름, 학번 출력

    do{//do-while문을 통하여 반복 출력
        printf("\n\n");
        printf("--------------------------------------------------------------------\n");
        printf("                  Binary Search Tree #2                             \n");
        printf("--------------------------------------------------------------------\n");
        printf(" Initialize BST       = z                                          \n");
        printf(" Insert Node          = i      Delete Node                 = d \n");
        printf(" Recursive Inorder    = r      Iterative Inorder (Stack)   = t \n");
        printf(" Level Order (Queue)  = l      Quit                        = q \n");
        printf("--------------------------------------------------------------------\n");
        //메뉴 출력

        printf("Command = ");
        scanf(" %c", &command);
        //명령어 입력받기

        switch(command) {//스위치문
        //소문자와 대문자 모두 입력 받기
        case 'z': case 'Z'://z키를 입력 받았을 때
            initializeBST(&head);//initializeBST함수 실행
            break;//스위치문 끝
        case 'q': case 'Q'://q키를 입력 받았을 때
            freeBST(head);//freeBST함수 실행
            break;//스위치문 끝
        case 'i': case 'I'://i키를 입력 받았을 때
            printf("Your Key = ");
            scanf("%d", &key);
            //키 값 입력받기
            insert(head, key);//insert함수 실행
            break;//스위치문 끝
        case 'd': case 'D'://d키를 입력 받았을 때
            printf("Your Key = ");
            scanf("%d", &key);
            //키 값 입력받기
            deleteNode(head, key);
            //deleteLeafNode함수 실행
            break;//스위치문 끝

        case 'r': case 'R'://r키를 입력 받았을 때
```

```
            recursiveInorder(head->left);
            //recursiveInorder함수 실행
            break;//스위치문 끝
        case 't': case 'T'://t키를 입력 받았을 때
            iterativeInorder(head->left);
            //iterativeInorder함수 실행
            break;//스위치문 끝

        case 'l': case 'L'://l키를 입력 받았을 때
            levelOrder(head->left);
            //levelOrder함수 실행
            break;//스위치문 끝

        case 'p': case 'P'://p키를 입력 받았을 때
            printStack();
            //printStack함수 실행
            break;//스위치문 끝

        default://이 외의 입력을 받았을 때
            printf("\n       >>>>>   Concentration!!   <<<<<     \n");
            //경고문 출력
            break;//스위치문 끝
        }

    }while(command != 'q' && command != 'Q');
    //q키를 사용하기 전까지 계속 반복
    return 1;//종료
}

int initializeBST(Node** h) {//BST초기화함수

    /* if the tree is not empty, then remove all allocated nodes from the tree*/
    if(*h != NULL)//h가 NULL이 아니라면
        freeBST(*h);//freeBST함수 실행

    /* create a head node */
    *h = (Node*)malloc(sizeof(Node));
    //h를 동적 메모리 할당
    (*h)->left = NULL;  /* root */
    //h의 left값을 NULL로 초기화
    (*h)->right = *h;
    //h의 right에 h를 대입
    (*h)->key = -9999;
    //h의 key값을 -9999로 초기화

    top = -1;
    //top에 -1대입

    front = rear = -1;
```

```
148     //-1을 rear에 삽입후 front에 삽입
149
150     return 1;//종료
151 }
152
153
154
155 void recursiveInorder(Node* ptr)
156 //재귀적 중위 순회 함수
157 {
158     if(ptr) {//ptr이라면
159         recursiveInorder(ptr->left);
160         //recursiveInorder함수 실행, left
161         printf(" [%d] ", ptr->key);
162         //ptr의 키 값 출력
163         recursiveInorder(ptr->right);
164         //recursiveInorder함수 실행, right
165     }
166 }
167
168 /**
169  * textbook: p 224
170  */
171 void iterativeInorder(Node* node)
172 //반복 중위 순회 함수
173 {
174     for(;;)//for 문의 무한루프
175     {
176         for(; node; node = node->left)
177             push(node);
178             //push 함수 실행
179         node = pop();
180         //pop함수 실행후 node에 삽입
181
182         if(!node) break;//node가 아니라면 중지
183         printf(" [%d] ", node->key);
184         //node의 key값 출력
185
186         node = node->right;
187         //node의 right 값을 node에 삽입
188     }
189 }
190
191 /**
192  * textbook: p 225
193  */
194 void levelOrder(Node* ptr)
195 //노드 레벨 순으로 순회하는 함수
196 {
```

```
197     // int front = rear = -1;
198
199     if(!ptr) return; /* empty tree */
200     //ptr이 아니라면 종료
201
202     enQueue(ptr);
203
204     for(;;)//반복문
205     {
206         ptr = deQueue();//deQueue값 ptr에 대입
207         if(ptr) {//ptr 이라면
208             printf(" [%d] ", ptr->key);
209             //ptr의 key값 출력
210
211             if(ptr->left)
212             //ptr의 left값이라면
213                 enQueue(ptr->left);
214                 //enqueue함수 실행, left
215             if(ptr->right)
216             //ptr의 right값이라면
217                 enQueue(ptr->right);
218                 //enqueue함수 실행, right
219         }
220         else//이외
221             break;//종료
222
223     }
224
225 }
226
227
228 int insert(Node* head, int key)
229 //삽입 함수
230 {
231     Node* newNode = (Node*)malloc(sizeof(Node));
232     //node 포인터형 변수 newNode 선언후 동적메모리할당
233     newNode->key = key;
234     //newnode의 key값을 key값으로 초기화
235     newNode->left = NULL;
236     //newnode의 left값을 NULL로 초기화
237     newNode->right = NULL;
238     //newnode의 right값을 NULL로 초기화
239
240     if (head->left == NULL) {
241     //head의 left값이 NULL이라면
242         head->left = newNode;
243         //head의 left값에 newNOde값 대입
244         return 1;//종료
245     }
```

```
246
247     /* head->left is the root */
248     Node* ptr = head->left;
249     //node 포인터변수 ptr선언 후 head의 left값 대입
250
251     Node* parentNode = NULL;
252     //node 포인터변수 parentNode선언 후 NULL로 초기화
253     while(ptr != NULL) {
254     //ptr이 NULL과 다르다면
255
256         /* if there is a node for the key, then just return */
257         if(ptr->key == key) return 1;
258         //ptr의 key값이 key값과 같다면 종료
259
260         /* we have to move onto children nodes,
261          * keep tracking the parent using parentNode */
262         parentNode = ptr;
263         //parentNode에 ptr값 대입
264
265         /* key comparison, if current node's key is greater than input key
266          * then the new node has to be inserted into the right subtree;
267          * otherwise the left subtree.
268          */
269         if(ptr->key < key)
270         //ptr의 key값이 key보다 작다면
271             ptr = ptr->right;
272             //ptr의 right값을 ptr에 대입
273         else//이외
274             ptr = ptr->left;
275             //ptr의 left값을 ptr에 대입
276     }
277
278     /* linking the new node to the parent */
279     if(parentNode->key > key)
280     //parentNode의 key값이 key보다 크다면
281         parentNode->left = newNode;
282         //parentNode의 left에 newNode 대입
283     else//이외
284         parentNode->right = newNode;
285         //parentNode의 right에 newNode 대입
286     return 1;//종료
287 }
288
289
290 int deleteNode(Node* head, int key)
291 //노드 삭제 함수
292 {
293     if (head == NULL) {
294     //head가 NULL이라면
```

```
295         printf("\n Nothing to delete!!\n");
296         //삭제할 것이 없음을 알림
297         return -1;//종료(비정상)
298     }
299 
300     if (head->left == NULL) {
301     //head의 left값이 NULL이라면
302         printf("\n Nothing to delete!!\n");
303         //삭제할 것이 없음을 알림
304         return -1;//종료(비정상)
305     }
306 
307     /* head->left is the root */
308     Node* root = head->left;
309     //node포인트변수 root선언 후 head의 left값 대입
310 
311     Node* parent = NULL;
312     //node포인트변수 parent선언 후 NULL로 초기화
313     Node* ptr = root;
314     //node포인트변수 ptr선언 후 root값 대입
315 
316     while((ptr != NULL)&&(ptr->key != key)) {
317     //ptr이 NULL이랑 다르고, ptr의 key값이 key값가 다르다면
318         if(ptr->key != key) {
319         //ptr의 key값이 key값과 다르다면
320 
321             parent = ptr;   /* save the parent */
322             //ptr을 parent에 대입
323 
324             if(ptr->key > key)
325             //ptr의 key값이 key보다 크다면
326                 ptr = ptr->left;
327                 //ptr의 left값을 ptr에 대입
328             else//이외
329                 ptr = ptr->right;
330                 //ptr의 right값을 ptr에 대입
331         }
332     }
333 
334     /* there is no node for the key */
335     if(ptr == NULL)//ptr이 NULL이라면
336     {
337         printf("No node for key [%d]\n ", key);
338         //key의 노드가 없음을 출력
339         return -1;
340         //종료
341     }
342 
343     /*
```

```
344         * case 1: the node which has to be removed is a leaf node
345         */
346        if(ptr->left == NULL && ptr->right == NULL)
347        //ptr의 left값이 NULL이고, ptr의 right값도 NULL이라면
348        {
349            if(parent != NULL) { /* parent exists, parent's left and right links are adjusted */
350            //parent가 NULL이랑 다르다면
351                if(parent->left == ptr)
352                //parent의 left값이 ptr이랑 같다면
353                    parent->left = NULL;
354                    //parent의 left값을 NULL로 초기화
355
356                else//이외
357                    parent->right = NULL;
358                    //parent의 right값을 NULL로 초기화
359            } else {//이외
360                /* parent is null, which means the node to be deleted is the root */
361                head->left = NULL;
362                //head의 left값을 NULL로 초기화
363
364            }
365
366            free(ptr);//ptr 할당해제
367            return 1;//종료
368        }
369
370        /**
371         * case 2: if the node to be deleted has one child
372         */
373        if ((ptr->left == NULL || ptr->right == NULL))
374        //ptr의 left값이 NULL이거나 ptr의 right값이 NULL이라면
375        {
376            Node* child;
377            //node 포인터변수 child 선언
378            if (ptr->left != NULL)
379            //ptr의 left의 값이 NULL이랑 다르다면
380                child = ptr->left;
381                //ptr의 left값을 child에 대입
382            else //이외
383                child = ptr->right;
384                //ptr의 right값을 child에 대입
385
386            if(parent != NULL)
387            //parent가 NULL이 아니라면
388            {
389                if(parent->left == ptr)
390                //parent의 left값이 ptr이랑 같다면
391                    parent->left = child;
392                    //child값을 parent의 left에 대입
```

```
393             else//아외
394                 parent->right = child;
395                 //child값을 parent의 right에 대입
396         } else {//이 외
397             /* parent is null, which means the node to be deleted is the root
398              * and the root has one child. Therefore, the child should be the root
399              */
400             root = child;//child를 root에 삽입
401         }
402 
403         free(ptr);//ptr 할당해제
404         return 1;//종료
405     }
406 
407     /**
408      * case 3: the node (ptr) has two children
409      *
410      * we have to find either the biggest descendant node in the left subtree of the ptr
411      * or the smallest descendant in the right subtree of the ptr.
412      *
413      * we will find the smallest descendant from the right subtree of the ptr.
414      *
415      */
416 
417     Node* candidate;
418     //Node 포인트변수 candidate 선언
419     parent = ptr;//ptr을 parent에 대입
420 
421 
422     candidate = ptr->right;
423     //ptr의 right값을 candidate에 대입
424 
425     /* the smallest node is left deepest node in the right subtree of the ptr */
426     while(candidate->left != NULL)
427     //candidate의 left값이 NULL과 다르다면
428     {
429         parent = candidate;
430         //candidate값을 parent에 대입
431         candidate = candidate->left;
432         //candidate의 left값을 candidate에 대입
433     }
434 
435     /* the candidate node is the right node which has to be deleted.
436      * note that candidate's left is null
437      */
438     if (parent->right == candidate)
439     //parent의 right와 candidate와 같다면
440         parent->right = candidate->right;
441         //candidate의 right값을 parent의 right에 대입
```

```
442         else//이외
443             parent->left = candidate->right;
444             //candidate의 right값을 parent의 left에 대입
445 
446         /* instead of removing ptr, we just change the key of ptr
447          * with the key of candidate node and remove the candidate node
448          */
449 
450         ptr->key = candidate->key;//candidate의 key값을 ptr의 key에 대입
451 
452         free(candidate);//candidate 할당해제
453         return 1;//종료
454     }
455 
456 
457     void freeNode(Node* ptr)
458     //freeNode 함수
459     {
460         if(ptr) {//ptr이라면
461             freeNode(ptr->left);
462             freeNode(ptr->right);
463             //freeNode함수 실행, left, right
464             free(ptr);//ptr 할당해제
465         }
466     }
467 
468     int freeBST(Node* head)//freeBST함수
469     //BST: 이진 탐색 트리
470     {
471 
472         if(head->left == head)
473         //head의 값을 head의 left에 대입
474         {
475             free(head);//head 할당해제
476             return 1;//종료
477         }
478 
479         Node* p = head->left;
480         //포인트 변수 node p선언후 head의 left값 대입
481 
482         freeNode(p);//freenode함수 실행
483 
484         free(head);//head 할당해제
485         return 1;//종료
486     }
```

```
> Executing task: cmd /C c:\Users\진유성\Desktop\Example\.vscode\bst-2-full <

[----- [Yuseong Jin] [2018038022] -----]

------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = z



------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = i
Your Key = 10



------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = i
Your Key = 20



------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = i
Your Key = 15
```

```
------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = r
 [10]  [15]  [20]

------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = l
 [10]  [20]  [15]

------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = t
 [10]  [15]  [20]

------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = d
Your Key = 15


------------------------------------------------------------
                  Binary Search Tree #2
------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = i      Delete Node                 = d
 Recursive Inorder    = r      Iterative Inorder (Stack)   = t
 Level Order (Queue)  = l      Quit                        = q
------------------------------------------------------------
Command = q
터미널 프로세스 "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -Command cmd /C c:\Users\진유성\Desktop
```