EDDI
Electronic Design
Development Institute

에디로봇아카데미
임베디드 마스터 Lv2 과정

제 1기

2021. 10. 29

김태훈

과제 Point

EDDI
Electronic Design Development Institute

Single pointer와 Double pointer의 구현 시 큰 특징 2가지

1. Double Pointer은 function stack frame에서 main stack frame에 접근이 가능하기 때문에 return 값이 필요 없지만
Single Pointer는 return이 필요함.

2. Double Pointer와 Single Pointer의 비교 시 가장 두드러진 차이를 보이는 부분이 Tree 구현할 때인데, tree는 2개의 pointer를 가지고 있기 때문이다.
Double pointer을 사용할 경우 이전에 접근했던 pointer의 주소(left or right)를 가지고 있기 때문에 코드가 깔끔해지는 반면
Single pointer는 left나 right의 주소를 받을 수 없기 때문에 내가 왼쪽으로 진입 했는 여부를 기록하는 flag variable하고, backup 변수가 필요하다.

STRUCT data structure

EDDI
Electronic Design Development Institute

```c
typedef struct _stack stack;
typedef struct _queue queue;
typedef struct _tree tree;

struct _stack
{
    int data;
    struct _stack* link;
};

struct _queue
{
    int data;
    struct _queue* link;
};

struct _tree
{
    int data;
    struct _tree* left;
    struct _tree* right;
};
```

CREATE data structure

EDDI
Electronic Design Development Institute

```c
stack* create_new_stack_node()
{
    stack* tmp = malloc(sizeof(stack));
    tmp->link = NULL;
    return tmp;
}

queue* create_new_queue_node()
{
    queue* tmp = malloc(sizeof(queue));
    tmp->link = NULL;
    return tmp;
}

tree* create_new_tree_node()
{
    tree* tmp = malloc(sizeof(tree));
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}
```

STACK

EDDI
Electronic Design
Development Institute

```
stack* push_stack_data(stack* root, int data)
{
    stack* tmp = root;
    root = create_new_stack_node();
    root->data = data;
    root->link = tmp;
    return root;
}

stack* pop_stack_data(stack* root)
{
    stack* tmp = root;
    root = root->link;
    free(tmp);
    return root;
}
```

INSERT QUEUE

EDDI
Electronic Design Development Institute

```
queue* insert_queue_data(queue* root, int data)
{
    queue* tmp = root;

    if(!root)
    {
        root = create_new_queue_node();
        root->data = data;
        return root;
    }
    else
    {
        while(root->link)
        {
            root = root->link;
        }
        root->link = create_new_queue_node();
        root->link->data = data;
        return tmp;
    }
}

queue* delete_queue_data(queue* root)
{
    printf("deletequeue\n");
    queue* tmp = root;
    root = root->link;
    free(tmp);
    return root;
}
```

사실 queue와 stack은
Double pointer나 single pointer나
구현 상에 큰 차이점을 느끼기는 힘들다.

INSERT TREE

EDDI
Electronic Design
Development Institute

```
tree* insert_tree_data(tree* root, int data)
{
    tree* tmp = root;


    if(!root)
    {
        root = create_new_tree_node();
        root->data = data;
        return root;
    }
    while(root)
    {
        if(root->data > data)
        {
            if(root->left)
                root = root->left;
            else
            {
                root->left = create_new_tree_node();
                root->left->data = data;
                return tmp;
            }

        }
        else if(root->data < data)
        {
            if(root->right)
                root = root->right;
            else
            {
                root->right = create_new_tree_node();
                root->right->data = data;
                return tmp;
            }
        }
    }
    return tmp;
}
```

DELETE NODE

EDDI
Electronic Design Development Institute

```
tree* delete_tree_data(tree* root, int data)
{
    tree* tmp = root;
    tree* prev_cursor = root;
    int left = 0;
    int flag_root = 1;
    if(!root)
    {
        printf("there is no %d in tree\n",data);
    }

    while(root->data !=  data)
    {
        if(root->data < data)
        {
            prev_cursor = root;
            root = root->right;
            left = 0;
            flag_root = 0;
        }
        else
        {
            prev_cursor = root;
            root = root->left;
            left = 1;
            flag_root = 0;
        }
        if(!root)
        {
            printf("there is no %d in tree\n", data);
            return tmp;
        }
    }
```

```
if(root->right && root->left)
{
    tree* cursor = root->right;
    tree* prev = root;
    while(cursor->left)
    {
        prev = cursor;
        cursor = cursor->left;
    }
    tree* tmp2 = cursor;
    root->data = cursor->data;
    if(prev == root)
        prev->right = cursor->right;
    else
        prev->left = cursor->right;
    free(tmp2);
}
else if(root->right)
{
    tree* tmp2 = root;
    if(flag_root)
    {
        prev_cursor = root->right;
        free(tmp2);
        return prev_cursor;
    }
    if(left)
        prev_cursor->left = root->right;
    else
        prev_cursor->right = root->right;
    free(tmp2);
}
```

DELETE TREE

EDDI
Electronic Design Development Institute

```
else
{
    free(root);
    if(flag_root)
        return NULL;
    if(left)
        prev_cursor->left = NULL;
    else
        prev_cursor->right = NULL;
}

return tmp;
```

구현할 때 어려웠던 점
1) root가 가리키는 node를 삭제할 때
2) 왼쪽에서 왔는지 오른쪽에서 왔는지 구분자가 따로 추가해야 한다는 것
이 때문에 구현 시에 backup 변수들을 많이 추가할수록 구현이 편해진다.

Main 함수

```
int main()
{
    stack* root_stack = NULL;
    queue* root_queue = NULL;
    tree* root_tree = NULL;

    int i;

    int data[10] = {8,3,10,1,6,4,7,14,13,2};

    printf("///// stack /////\n");
    for(i=0;i<10;i++)
        root_stack = push_stack_data(root_stack, data[i]);
    printf("after push\n");
    print_stack_data(root_stack);
    for(i=0;i<10;i++)
        root_stack = pop_stack_data(root_stack);
    printf("after pop\n");
    print_stack_data(root_stack);


    printf("///// queue /////\n");
    for(i=0;i<10;i++)
        root_queue = insert_queue_data(root_queue, data[i]);
    printf("after insertion\n");
    print_queue_data(root_queue);
    for(i=0;i<10;i++)
        root_queue = delete_queue_data(root_queue);
    printf("after deletion\n");
    print_queue_data(root_queue);


    printf("///// tree /////");
        for(i=0;i<10;i++)
            root_tree = insert_tree_data(root_tree, data[i]);
        printf("after insertion\n");

        print_tree_data(root_tree);
#if 1
        for(i=9;i>=0;i--)
    {
        printf("delete %d\n",data[i]);
            root_tree = delete_tree_data(root_tree, data[i]);
        print_tree_data(root_tree);
    }
```

빨간 네모 칸과 같이 main함수에서 Function의 return값을 포인터로 받아야 갱신이 된다.
이 점이 double pointer와 Single pointer의 차이 중 하나.