자료구조 HW2

2016124145 양해찬

1.

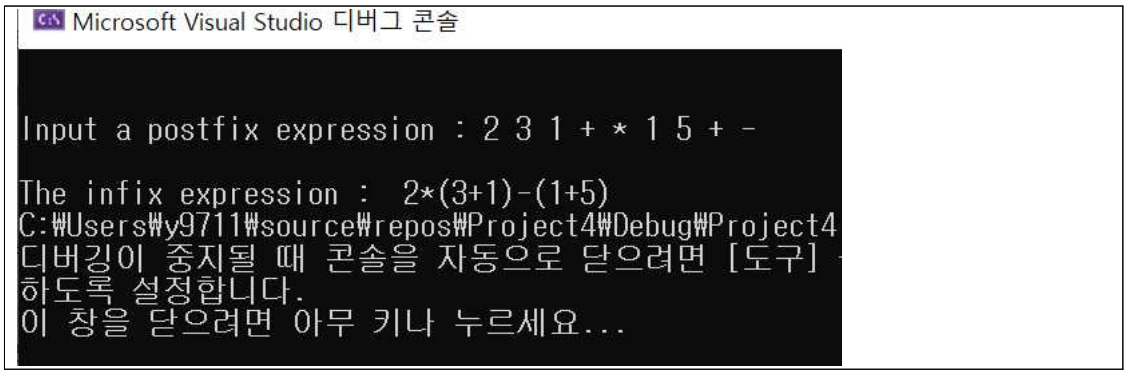
```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int stack[MAX];
int top;
typedef struct _node {
    int key;
    struct _node* left;
    struct _node* right;
} node;//트리 노드를 위한 구조체 선언
node* head, * tail;
void init_stack(void) {
    top = -1;
}
void visit(node* t) {
    printf("%c", t->key);
}//visit시 해당 노드의 key출력
int push(int t) {
    if (top >= MAX - 1) {
        printf("\n Stack overflow.");
        return -1;
    }
    stack[++top] = t;
    return t;
}
int pop(void) {
    if (top < 0) {
        printf("\n Stack underflow.");
        return -1;
    }
    return stack[top--];
}
int is_operator(int k) {
    return (k == '+' || k == '-' || k == '*' || k == '/');
}
int precedence(int op) {
    if (op == '+') return 0;
    if (op == '-') return 1;
```

```
    if (op == '*' || op == '/') return 2;
    else return 3;
}// 과제에선 -에도 괄호를 쳐주기 때문에 우선순위 * = / > - > + 순서로 설정
void init_tree(void)
{
    head = (node*)malloc(sizeof(node));
    tail = (node*)malloc(sizeof(node));
    head->left = tail;
    head->right = tail;
    tail->left = tail;
    tail->right = tail;
}   //트리생성
node* make_parse_tree(char* p) {
    node* t;
    while (*p) {
        while (*p == ' ') p++;
        t = (node*)malloc(sizeof(node));
        t->key = *p;
        t->left = tail;
        t->right = tail;
        if (is_operator(*p)) {
            t->right = pop();
            t->left = pop();
        }
        push(t);
        p++;
    }
    return pop();
}//수식나무 생성
void brackets(node* t) {//괄호생성함수
    node* l = t;//괄호를 칠 연산노드의 왼쪽을 위한 변수
    node* r = t;//괄호를 칠 연산노드의 오른쪽을 위한 변수
    if (t != tail) {
        brackets(t->left);//현재 노드의 왼쪽부분 수행
        if (is_operator(t->key) && is_operator(t->right->key) && precede
nce(t->right->key) < precedence(t->key)) {
            //오른쪽 child 노드의 연산이 현재 노드 보다 우선 순위가 낮으면
            l = r = t->right;//오른쪽 연산에 괄호가 필요하므로 l r 오른쪽으
```

```
로 이동
            node* lb = (node*)malloc(sizeof(node));
            node* rb = (node*)malloc(sizeof(node));//(을 위한 lb, )을 위
한 rb
            while (l->left != tail) {//tail전까지
                l = l->left;//l 이동
            }
            l->left = lb;
            lb->left = tail;
            lb->right = tail;
            lb->key = '(';//왼쪽 마지막에 ( 노드 만들어줌
            while (r->right != tail) {//tail 전까지
                r = r->right; //r 이동
            }
            r->right = rb;
            rb->left = tail;
            rb->right = tail;
            rb->key = ')';      //오른쪽 마지막에 ) 노드 생성
            free(lb);
            free(rb);
        }
        else if (is_operator(t->key) && is_operator(t->left->key) && pre
cedence(t->left->key) <= precedence(t->key)) {
            //왼쪽 child 노드의 연산의 우선 순위가 현재 노드보다 높지 않으면
            l = r = t->left;    //왼쪽 연산에 괄호가 필요하므로 l r 왼쪽으로
이동
            node* lb = (node*)malloc(sizeof(node));
            node* rb = (node*)malloc(sizeof(node));
            while (l->left != tail) {//이하 동일
                l = l->left;
            }
            l->left = lb;
            lb->left = tail;
            lb->right = tail;
            lb->key = '(';
            while (r->right != tail) {
                r = r->right;
            }
```

```
            r->right = rb;
            rb->left = tail;
            rb->right = tail;
            rb->key = ')';
            free(lb);
            free(rb);
        }
        brackets(t->right); //오른쪽에 대해서도 똑같이 수행
    }
}
void inorder_traverse(node* t) {//중위 표기법 탐색
    if (t != tail) {
        inorder_traverse(t->left);
        visit(t);
        inorder_traverse(t->right);
    }
}
void main(void) {
    char post[256];
    init_stack();//스택생성
    init_tree();//트리생성
    printf("\n\nInput a postfix expression : ");
    gets(post);//후위표기법 입력
    if (*post == NULL) {
        printf("\n Program ends...");
        exit(0);
    }

    head->right = make_parse_tree(post);//수식나무를 헤드노드와 ㅗ연결
    brackets(head->right);              //수식나무에 괄호 생성
    printf("\nThe infix expression :  ");
    inorder_traverse(head->right);//수식나무 중위표기법 탐색
}
```



2.

```c
#define _CRT_SECURE_NO_WARNINGS
#define BASE(i)     ((char*)base + (i)*width)
#include<stdio.h>
#include<stdlib.h>
typedef int (*FCMP)(const void*, const void*);
typedef struct _people {
    char major[10];
    char name[15];
    double point;
}people;
int major_cmp(const void* a, const void* b) {
    if (((people*)a)->major == ((people*)b)->major) {//학과가 같으면
        return strcmp(((people*)a)->name, ((people*)b)->name);
        //이름으로 비교
    }
    return strcmp(((people*)a)->major, ((people*)b)->major);
}//구조체 데이터를 학과로 비교
int point_cmp(const void* a, const void* b) {
    return (((people*)b)->point - ((people*)a)->point);
}//구조체 데이터를 점수로 비교
int split(void* base, size_t nelem, size_t width, FCMP fcmp) {
    int i, j;
```

```
    void* v, * t;
    v = malloc(width);
    v = BASE(nelem - 1);//v = a[n-1]
    t = malloc(width);
    i = -1;
    j = nelem - 1;
    while (1) {
        while (fcmp(BASE(++i), v) < 0);//while (a[++i] < v)
        while (fcmp(BASE(--j), v) > 0 && j > i);//while (a[--j] > v && j
> i)
        if (i >= j) break;
        memcpy(t, BASE(i), width);
        memcpy(BASE(i), BASE(j), width);
        memcpy(BASE(j), t, width);//t = a[i]; a[i] = a[ j]; a[j] = t
    }
    memcpy(t, BASE(i), width);
    memcpy(BASE(i), BASE(nelem - 1), width);
    memcpy(BASE(nelem - 1), t, width);//t = a[i]; a[i] = a[n-1]; a[n-1]
= t
    return i;
    free(t);
    free(v);
}
void quick_sort(void* base, size_t nelem, size_t width, FCMP fcmp) {
    int i;
    if (nelem > 1) {
        i = split(base, nelem, width, fcmp);//i = split(a, n)
        quick_sort(base, i, width, fcmp);//quick_sort(a, i)
        quick_sort(BASE(i + 1), nelem - i - 1, width, fcmp);//quick_sort
(a+i+1, n-i-1)
    }
}

int main() {
    people p[5];//구조체 변수 선언
    int i;
    FILE* fp = fopen("infile.txt", "r"); //구조체로 입력할 txt파일 fopen
    for (i = 0;i < 5;i++) {//해당 파일에 대해 구조체 배열에 각각 저장
```

```
        fscanf(fp, "%s %s %Lf\n", &p[i].major, &p[i].name, &p[i].point);
    }
    fclose(fp);
    FILE* fp1 = fopen("result.txt", "w");//정렬 후 저장할 파일
    fprintf(fp1, "-------------------------\n");
    quick_sort(p, 5, sizeof(people), major_cmp);//전공에 대해 퀵정렬
    for (i = 0;i < 5;i++) {//정렬한 구조체 배열을 파일에 써줌
        fprintf(fp1, "\n%s %s %.1Lf\n", p[i].major, p[i].name, p[i].poin
t);
    }
    fprintf(fp1, "\n-------------------------\n");
    quick_sort(p, 5, sizeof(people), point_cmp);//점수에 대해 퀵정렬
    for (i = 0;i < 5;i++) {//정렬한 구조체 배열을 파일에 써줌
        fprintf(fp1, "\n%s %s %.1Lf\n", p[i].major, p[i].name, p[i].poin
t);
    }
    fprintf(fp1, "\n-------------------------\n");
    fclose(fp1);
}
```