자료구조 HW2 카드 짝짓기

CSE3080

20231604 정유연

1. stack 구조체 코드

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning(disable:4996)

typedef struct element {
    char key;
}Element;

typedef struct stack {
    Element data;
    struct stack* link;
}Stack;
```

- typedef struct element, typedef struct stack

  element와 stack포인터(self referential) link가 변수로 있다. element는 해당 원소의 값을 저장하는 element이고, link는 다음 노드(스택)을 가리키는 포인터이다.

```c
void push(Stack** top, Element item) {
    Stack* tmp = (Stack*)malloc(sizeof(Stack));
    tmp->data = item;
    tmp->link = *top;
    *top = tmp;
}
```

- void push(Stack** top, Element item)

  스택에 새로운 값을 푸시하는 함수이다. 후입 선출의 구조를 가지는 스택에 맞게 새로운 값은 top에 쌓는다.

  새로운 스택 포인터를 만들어 Stack의 크기만큼 동적할당하고 item의 값을 저장하고 top을 다음 노드로 link한다. 이렇게 새롭게 만들어진 스택이 top이 된다.

```
void pop(Stack** top) {
    Stack* tmp = *top;
    *top = tmp->link;
    free(tmp);
}
```

- Element pop(Stack** top)

  맨 위 원소를 지우는 pop함수이다. 스택 이중 포인터를 매개변수로 받아서 top의 위치를 바꾸고 기존의 맨 위 스택(top)은 메모리 해제 free시킨다.

```
void printStack(Stack** top) {
    if (*top == NULL) return;
    Stack* current = *top;
    printf("HELLO?\n");
    while (current->link) {
        printf("%c->", current->data.key);
        current = current->link;
    }
    printf("%c\n", current->data.key);
}
```

- void printStack(Stack** top)

  스택을 출력하는 함수다. 스택 포인터 current를 설정해서 맨 위 스택을 출력하고 다음 스택으로 넘어가는 과정을 반복한다.

2. 괄호 검사 check 함수

```
int check(Stack** top) {
    Stack* current = *top;
    Stack* previous = NULL;
    while (*top != NULL) {
        current = *top;
        previous = NULL;
```

- 이번 코드의 핵심 부분인 괄호 검사를 진행하는 함수다. 스택 이중 포인터를 매개변수로 받아서 메인 함수의 stack에 손상이 가지 않도록 한다.

  현재 스택을 가리키는 포인터인 current와 이전 노드를 가리키는 포인터인 previous를 선언한다. top이 널을 가리킬 때까지 즉 스택의 모든 노드를 검사할 때까지 아래 계산을 반복한다. while문을 이용했다.

```
        if (current->data.key == ')') {
            while (current->data.key != '(') {
                previous = current;
                current = current->link;
                if (current == NULL) {
                    return 0;
                }
            }
            previous->link = current->link;
            pop(&(*top));
            free(current);
        }
```

- 스택 구조체의 특성상 current는 맨 위 스택인 맨 뒤 글자부터 가리키기 때문에 닫는 괄호 ') ' 뒤에 여는 괄호 ' ('가 나오도록 프로그래밍 한다.
- current의 값이 닫는 괄호 ') ' 라면 여는 괄호 ' ('가 나올 때까지 while문으로 다음 노드를 탐색한다. current가 스택의 가장 끝까지 갔을 때도 닫는 괄호를 찾지 못한다면 0을 반환한다.
- 위 while문을 벗어났다면 current의 값이 ' ('이므로 괄호의 짝이 완성되었다. 다음 검사를 진행하기 위해 검사를 통과한 맨 위 노드이자(top) 닫는 괄호 ') '을 pop해준다. current위치에 있는 여는 괄호 ' ('는 노드의 연결을 배제하고 free 시킴으로써 여는 괄호와 닫는 괄호 모두 스택에서 지운다.

```
        else if (current->data.key == ']') {
            while (current->data.key != '[') {
                previous = current;
                current = current->link;
                if (current == NULL) {
                    return 0;
                }
            }
            previous->link = current->link;
            pop(&(*top));
            free(current);
        }
        else return 0;
    }
    return 1;
}
```

- 대괄호 '] ' 일 때도 같은 작업을 해준다. else는 닫는 괄호) 이나] 이가 아닌 여는 괄호가 먼저 나온다면 조건을 만족하지 않으므로 바로 0을 반환한다. 전체 while문(top!=NULL)을 벗어났다면 모든 조건을 만족하는 문자열이므로 1을 반환한다.

3. 메인함수

```c
int main() {
    char str[21];
    Stack* top = NULL;
    scanf("%s", str);

    for (int i = 0; i < strlen(str); i++) {
        Element item = { str[i] };
        push(&top, item);
    }
    printf("%d", check(&top));

    while (top!=NULL) {
        pop(&top);
    }

    return 0;
}
```

- 문자열을 입력받는다. 문제에서 카드는 최대 20장이 주어진다 했으므로 char str[21]로 먼저 메모리를 할당한다.
- 연결 리스트 기반으로 된 스택을 활용해야 하므로 위에 만들어둔 stack구조체에 문자열의 character을 하나씩 push해준다.
- stack을 check함수에 넣어 검사한 결과를 print한다.
- 메모리 누수를 막기 위해 나머지 스택 top을 pop 시킨다. 조건을 만족하는 스택이라면 check 함수에서 모두 free가 됐겠지만, 조건을 만족하지 않는 문자열은 조건 불만족 즉시 함수를 빠져나가기 때문에 free되지 않은 채 메모리에 남아있을 것이다. 이를 위해 나머지도 pop한다.

4. 프로그램 실행 결과