Cykor 1주차
Call stack 시뮬레이터 구현 보고서

학번 : 2025350213
이름 : 승창민

1. 과제 목적

본 과제는 C언어에서 함수 호출 시의 Call stack 동작 원리를 직접 구현함으로서 실제 시스템에서의 스택 프레임 구성, 함수 호출/복귀 과정, SP/FP의 역할을 이해하는 것을 목표로 한다.

2. 프로그램 발전 단계

[1]

1) 강의에서 준 예제를 이용하여 push, pop 함수를 구현
2) 함수의 큰 흐름을 확인하고, 알고리즘을 이해
   함수 호출 순서는 이하와 같음.
   main -> func 1 -> func 2 -> func 3 -> func 2 -> func 1 -> main
3) push 함수 설계에 있어, 인덱스와 인덱스 정보를 동시에 전달받을지, 개별로 받을지 고민.
   'push(index, info)' 형식으로 패러미터를 전달받도록 함수를 형성하면 메인 코드의 길이를 단축할 수 있겠다고 판단.
4) 위 내용들을 바탕으로 기본적인 call stack 구현

[2]

1) input을 보다 유동적으로 받아보기로 함
   코드를 건드려 가며 input이 진행되는 흐름을 더 자세히 파악함
2) 각 func를 시행할 때마다 새로 input을 받는 코드를 생성
3) input 시점을 조절할 수 있게 된 만큼 함수를 단계적으로 call, release 하도록 만들어보겠다는 새로운 목표 수립.

[3]

1) 초과 입력(excess input)이나 불충분 입력(insufficient input)이 들어오는 경우 오류가 발생함을 확인
2) scanf의 개수를 확인하는 코드를 이용해 경고문을 출력하려고 했으나 실패
   불충분 입력의 경우 정상적으로 경고문이 출력됨.
   하지만 초과 입력의 경우에는 문제가 해결되지 않음

3) 초과 입력 문제 해결 방안으로 getchar()함수 발견
 getchar() 함수를 이용하면 범위 외에 들어온 초과 입력값을 삭제할 수 있음.
 이 경우 초과된 입력값들을 자동으로 삭제하기 때문에 오류가 발생하지는 않음.
 하지만 의도한 대로 경고문이 나타나지는 않음.

[4]
1) input_clear_check 함수를 만듬
 정해진 길이까지 왔음에도 띄어쓰기가 아닌 별도의 입력값이 들어오는 경우, 0을 리턴하는 검사 함수임.
2) 이를 이용해 정상 입력이 들어올 때만 실행되는 검사 코드 구현
3) 이를 발전시켜 정상 입력값이 들어올 때까지 작동하는 무한 루프를 구현함.

3. 구현 개요
call_stack[] 배열과 stack_info[][] 배열을 사용하여 함수 호출 시의 스택 구조를 시뮬레이션
SP, FP를 직접 관리하며, 함수 호출 시 스택 프레임 구성 및 해제를 구현

push(), pop() 함수로 스택 조작 수행
print_stack()을 통해 현재 스택 상태 시각화
input_clear_check()를 통해 초과 입력 여부 확인
이후 입력값 오류 처리를 통해 잘못된 입력 차단

4. 함수별 역할
-main
사용자로부터 시작 명령과 초기 인자 3개 입력
입력값 유효성 검사 후 func 1 호출
-func 1
3개의 index, SFP, Return Address, 지역변수 1개를 push
func 2 호출 유도 후, pop으로 자신의 프레임 해제
-func 2
2개의 인자, SFP, Return Address, 지역변수 1개 push
func 3 호출 유도 후, pop으로 프레임 해제
-func 3
1개 인자, Return Address, SFP, 지역변수 2개 push

-push
index와 index_info를 각각 call_stack과 stack_info에 push함
-pop
스택 최상단의 항목을 제거함

-input_clear_check
함수의 입력값이 유효한 입력인지 길이 초과 여부를 확인

5. 최종 완성 코드

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define STACK_SIZE 50

int       call_stack[STACK_SIZE];
char      stack_info[STACK_SIZE][20];

int SP = -1;
int FP = -1;

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);

int input_clear_check() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {
        if (c != ' ') {
            return 0;
        }
    }
    return 1;
}


void push(int value, char* info) {
    if (SP >= STACK_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    else {
        SP++;
        call_stack[SP] = value;
        strcpy(stack_info[SP], info);
    }
}


void pop() {
    call_stack[SP] = 0;
    stack_info[SP][0] = '\0';
    SP--;
}
```

```
void print_stack()
{
    if (SP == -1)
    {
        printf("Stack is empty.\n");
        return;
    }

    printf("====== Current Call Stack ======\n");

    for (int i = SP; i >= 0; i--)
    {
        if (call_stack[i] != -1)
            printf("%d : %s = %d", i, stack_info[i], call_stack[i]);
        else
            printf("%d : %s", i, stack_info[i]);

        if (i == SP)
            printf("      <=== [esp]\n");
        else if (i == FP)
            printf("      <=== [ebp]\n");
        else
            printf("\n");
    }
    printf("================================\n\n");
}

void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    push(FP, "func1 SFP");
    push(-1, "Return Address");
    FP = SP;
    push(var_1, "var_1");

    print_stack();

    int start = 0;
    printf("Press 2 to operate Func 2? : ");
    scanf("%d", &start);

    if (start == 2) {
        int index21, index22;
        while(1){
            printf("Enter two index : ");
            if (scanf("%d %d", &index21, &index22) == 2 && (input_clear_check())) {
                break;
            }
            printf("Input Error : You must enter two index\n");
            while (getchar() != '\n');
        }
```

```
            func2(index21, index22);
        }
        printf("Press 0 to release Func2 : ");
        scanf("%d", &start);
        if (start == 0) {
            int saved_fp = FP;
            FP = call_stack[saved_fp];
            for (int i = 0; i < 6; i++) {
                pop();
            }
        }
        print_stack();
}

void func2(int arg1, int arg2)
{
        int var_2 = 200;
        push(arg1, "arg1");
        push(arg2, "arg2");
        push(FP, "func2 SFP");
        push(-1, "Return Address");
        FP = SP;
        push(var_2, "var_2");

        print_stack();

        int start = 0;
        printf("Press 3 to operate Func 3? : ");
        scanf("%d", &start);
        if (start == 3) {
            int index31;
            while(1){
                printf("Enter a index : ");
                if (scanf("%d", &index31)  == 1 && (input_clear_check())) {
                    break;
                }
                printf("Input Error : You must enter one index\n");
                while (getchar() != '\n');
            }
            func3(index31);
        }

        printf("Press 0 to release Func3 : ");
        scanf("%d", &start);
        if (start == 0) {
            int saved_fp = FP;
            FP = call_stack[saved_fp];
            for (int i = 0; i < 5; i++) {
                pop();
            }
        }
        print_stack();
}
```

```
void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func3 SFP");
    FP = SP;
    push(var_3, "var_3");
    push(var_4, "var_4");

    print_stack();

}

int main()
{
    int start = 0;
    printf("Press 1 to operate Func 1 : ");
    scanf("%d", &start);
    if (start == 1) {
        int index11, index12, index13;

        while(1){
            printf("Enter three index : ");
            if (scanf("%d %d %d", &index11, &index12, &index13) == 3 && (input_clear_check())) {
                break;
            }
            printf("Input Error : You must enter three index\n");
            while (getchar() != '\n');
        }

        func1(index11, index12, index13);

        printf("Press 0 to release Func3 : ");
        scanf("%d", &start);
        if (start == 0) {
            int saved_fp = FP;
            FP = call_stack[saved_fp];
            for (int j = 0; j < 6; j++) {
                pop();
            }
        }
    }

    print_stack();
    return 0;
}
```

6. 실행 예시

```
Press 1 to operate Func 1 : 1
Enter three index : 12 13 14 15
Input Error : You must enter three index
Enter three index : 1 2 3
====== Current Call Stack ======
5 : var_1 = 100    <=== [esp]
4 : Return Address    <=== [ebp]
3 : func1 SFP
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================

Press 2 to operate Func 2? : 2
Enter two index : 4 5
====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : Return Address    <=== [ebp]
8 : func2 SFP = 4
7 : arg2 = 5
6 : arg1 = 4
5 : var_1 = 100
4 : Return Address
3 : func1 SFP
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================
```

7. 느낀 점

이번 과제를 통해 함수 호출 시 실제로 어떤 데이터들이 스택에 쌓이고 해제되는지 시각적으로 파악할 수 있었고, 프레임 포인터(FP)와 스택 포인터(SP)의 역할을 명확히 이해할 수 있었다. 또한, 어떤 상황에서 프로그램에 오류가 발생하는지를 확인함으로서 사용자 입력의 유효성 검사를 통해 실용적인 예외 처리의 필요성에 대해 생각해 볼 수 있었다.