시스템해킹 1주차

1차 제출본

처음에는 push 함수와 pop 함수를 다루는 것을 중점적으로 생각하였다. 주어진 코드에서 제시된 것은 main과 fun1,2,3 함수를 서로 호출하는 것만 있었기에 push 함수와 pop함수를 작성하는 것으로 시작했다.

push함수를 작성할 때 주의해야하는 것은 sp가 stack size보다 커지면 안된다는 것이다. 그러면 stack overflow가 일어나므로 이를 판단하는 if문을 넣었고, 값이 push됨에 따라 sp가 증가하는 코드를 만들었다.

그리고 pop함수를 작성할 때에는 sp가 초기값인 -1보다 작아질 수 없기에 시작부터 pop을 호출하여 underflow가 일어나 것을 방지하는 if문을 넣었고, sp가 -1보다 크다면 sp를 1 감소시키면서 자동적으로 그 위에 저장된 값이 삭제되도록 제작하였다.

현재 코드의 문제점은 push에 넣은 값이 지나치게 클 때 발생하는 stack overflow와 fp, 함수 프롤로그, 에필로그 부분이 제대로 구현되지 않았다는 점이다. 이 부분을 보완하여 2차 제출본을 작성할 계획이다.

2차 제출본

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에

    int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ============================================================



    ============================stack_info 저장 규칙=============
```

```
   매개 변수 / 지역 변수를 push할 경우       : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우              : "Return Address"
   ============================================================
*/
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int    call_stack[STACK_SIZE];       // Call Stack을 저장하는 배열
char   stack_info[STACK_SIZE][20];   // Call Stack 요소에 대한 설명을 저장하는 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

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

   printf("push: %s = %d (SP: %d)\n", info, value, SP);
}

void  pop() {
   if (SP < 0) {
      printf("ERROR\n");
      return;
   }
```

```
    SP--;
}

void func_prologue(const char* func_name) {
    push(FP, func_name);
    printf(stack_info[SP], "%s SFP", func_name);
    push(-1, "Return Address");
    FP = SP;
}

void func_epilogue(int local_var_count) {
    for (int i = 0; i < local_var_count; i++) {
        pop();
    }
    pop();

    int saved_fp = call_stack[SP];
    pop();
    FP = saved_fp;
}

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

/*
    현재 call_stack 전체를 출력합니다.
    해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
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");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    if (FP == SP) {
        push(-1, "Return Address");
    }
    else {
        push(-1, "Return Address");
        push(-1, "func1 SFP");
        FP = SP;
    }
    printf("func1 start \n");
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    int var_1 = 100;
    push(var_1, "var_1");

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
```

```
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func2(int arg1, int arg2)
{
    printf("func2 start \n");
    int var_2 = 200;
    push(var_2, "var_2");

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();

    pop();
}


void func3(int arg1)
{
    printf("func3 start \n");
    int var_3 = 300;
    int var_4 = 400;
    push(var_3, "var_3");
    push(var_4, "var_4");

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();

    pop();
    pop();
}
```

```
//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    printf("main start \n");
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    pop();
    return 0;
}
```

함수 에필로그와 함수 프롤로그의 기능을 구현하려고 했다. 하지만 esp와 ebp의 위치가 제대로 출력되지 않아 수정이 필요하다. 그리고 push와 pop함수도 여러번 입출력해도 작동되는지 테스트해보려고 했다. (func1 → func2 → func1과 같은 형식) 하지만 노트북이 고장나서 더이상 vs code를 실행할 수 없는 환경에 있기에 머리로 가상 시뮬레이션을 돌리기로 했다.

1차 가상 시뮬레이션

일단은 코드를 다시 호출하려고 했던 2차 제출의 상황을 구현해보기로 했다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 ㄷ
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제어

    int call_stack[]     : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ============================================================
```

```
   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우       : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우              : "Return Address"
   ============================================================
*/
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int    call_stack[STACK_SIZE];        // Call Stack을 저장하는 배열
char   stack_info[STACK_SIZE][20];    // Call Stack 요소에 대한 설명을 저장하는 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

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

   printf("push: %s = %d (SP: %d)\n", info, value, SP);
}

void  pop() {
   if (SP < 0) {
      printf("ERROR\n");
```

```
        return;
    }
    SP--;
}

void func_prologue(const char* func_name) {
    push(FP, func_name);
    printf(stack_info[SP], "%s SFP", func_name);
    push(-1, "Return Address");
    FP = SP;
}

void func_epilogue(int local_var_count) {
    for (int i = 0; i < local_var_count; i++) {
        pop();
    }
    pop();

    int saved_fp = call_stack[SP];
    pop();
    FP = saved_fp;
}

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

/*
    현재 call_stack 전체를 출력합니다.
    해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
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");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    if (FP == SP) {
        push(-1, "Return Address");
    }
    else {
        push(-1, "Return Address");
        push(-1, "func1 SFP");
        FP = SP;
    }
    printf("func1 start \n");
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    int var_1 = 100;
    push(var_1, "var_1");
```

```
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func2(int arg1, int arg2)
{
    printf("func2 start \n");
    int var_2 = 200;
    push(var_2, "var_2");

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    func1(3,4,19);

    print_stack();

    pop();
}


void func3(int arg1)
{
    printf("func3 start \n");
    int var_3 = 300;
    int var_4 = 400;
    push(var_3, "var_3");
    push(var_4, "var_4");

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
```

```
    func2(15, 32);
    print_stack();

    pop();
    pop();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    printf("main start \n");
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    pop();
    return 0;
}
```

위와 같이 이전 함수로 다시 되돌아가는 코드를 작성하였다. 하지만, 너무 단순한 변화를 하였고, 이미 불렀던 함수를 다시 호출하는 과정에서 무한 루프에 빠질 것 같다는 생각이 들었다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 ㄷ
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에

    int call_stack[]     : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ============================================================
```

```
   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우       : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우              : "Return Address"
   ============================================================
*/
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int    call_stack[STACK_SIZE];        // Call Stack을 저장하는 배열
char   stack_info[STACK_SIZE][20];    // Call Stack 요소에 대한 설명을 저장하는 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

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

   printf("push: %s = %d (SP: %d)\n", info, value, SP);
}

void  pop() {
   if (SP < 0) {
      printf("ERROR\n");
```

```
        return;
    }
    SP--;
}

void func_prologue(const char* func_name) {
    push(FP, func_name);
    printf(stack_info[SP], "%s SFP", func_name);
    push(-1, "Return Address");
    FP = SP;
}

void func_epilogue(int local_var_count) {
    for (int i = 0; i < local_var_count; i++) {
        pop();
    }
    pop();

    int saved_fp = call_stack[SP];
    pop();
    FP = saved_fp;
}

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

/*
    현재 call_stack 전체를 출력합니다.
    해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
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");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    if (FP == SP) {
        push(-1, "Return Address");
    }
    else {
        push(-1, "Return Address");
        push(-1, "func1 SFP");
        FP = SP;
    }
    printf("func1 start \n");
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    int var_1 = 100;
    push(var_1, "var_1");
```

```
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(17);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func2(int arg1, int arg2)
{
    printf("func2 start \n");
    int var_2 = 200;
    push(var_2, "var_2");

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();

    pop();
}


void func3(int arg1)
{
    printf("func3 start \n");
    int var_3 = 300;
    int var_4 = 400;
    push(var_3, "var_3");
    push(var_4, "var_4");

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();

    pop();
    pop();
```

```
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    printf("main start \n");
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    pop();
    return 0;
}
```

그래서 func3을 func1에서도, func2에서도 출력을 하는 형태로 무한 루프가 되는 것을 막고 여러번 출력되는 상황을 고려하였다.

2차 가상 시뮬레이션

2차 시뮬레이션에서는 함수 프롤로그와 함수 에필로그를 더 발전시키는 것을 목적으로 진행하였다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 ㄷ
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제어

    int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ============================================================
```

```
   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우       : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우              : "Return Address"
   ============================================================
*/
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int    call_stack[STACK_SIZE];        // Call Stack을 저장하는 배열
char   stack_info[STACK_SIZE][20];    // Call Stack 요소에 대한 설명을 저장하는 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

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

   printf("push: %s = %d (SP: %d)\n", info, value, SP);
}

void  pop() {
   if (SP < 0) {
      printf("ERROR\n");
```

```
        return;
    }
    SP--;
}

void func_prologue(const char* func_name) {
    push(FP, func_name);
    sprintf(stack_info[SP], "%s SFP", func_name);
    push(-1, "Return Address");
    FP = SP;
}

void func_epilogue(int local_var_count) {
    for (int i = 0; i < local_var_count; i++) {
        pop();
    }
    pop();

    int saved_fp = call_stack[SP];
    pop();
    FP = saved_fp;
}

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

/*
    현재 call_stack 전체를 출력합니다.
    해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
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");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    if (FP == SP) {
        push(-1, "Return Address");
    }
    else {
        push(-1, "Return Address");
        push(-1, "func1 SFP");
        FP = SP;
    }
    printf("func1 start \n");
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    int var_1 = 100;
    push(var_1, "var_1");
```

```
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func2(int arg1, int arg2)
{
    printf("func2 start \n");
    int var_2 = 200;
    push(var_2, "var_2");

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();

    pop();
}


void func3(int arg1)
{
    printf("func3 start \n");
    int var_3 = 300;
    int var_4 = 400;
    push(var_3, "var_3");
    push(var_4, "var_4");

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();

    pop();
    pop();
```

```
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    printf("main start \n");
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    pop();
    return 0;
}
```

printf로 값을 출력한다면 문제가 발생하기 쉽다는 내용을 읽은 적이 있어, 이를 바탕으로 해결책을 고민하였다. 그 결과 printf를 sprintf로 수정한다면 stack_info[SP] 내용을 정확히 출력할 수 있다는 것을 알게 되었다. (하지만 이를 실제로 디버깅할 방법이 없기에 노트북이 고쳐진다면 디버깅을 하며 확인할 예정이다.)

3차 가상 시뮬레이션

func에 값을 입력하는 것을 기존에 있었던 것을 출력하는 방식에서 변경하여 코드를 진행할 때마다 값을 실시간으로 입력받는 방식으로 변경하고자 한다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 ㄷ
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에

    int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ============================================================
```

```
   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우       : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우              : "Return Address"
   ============================================================
*/
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int    call_stack[STACK_SIZE];        // Call Stack을 저장하는 배열
char   stack_info[STACK_SIZE][20];    // Call Stack 요소에 대한 설명을 저장하는 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

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

   printf("push: %s = %d (SP: %d)\n", info, value, SP);
}

void  pop() {
   if (SP < 0) {
      printf("ERROR\n");
```

```
        return;
    }
    SP--;
}

void func_prologue(const char* func_name) {
    push(FP, func_name);
    sprintf(stack_info[SP], "%s SFP", func_name);
    push(-1, "Return Address");
    FP = SP;
}

void func_epilogue(int local_var_count) {
    for (int i = 0; i < local_var_count; i++) {
        pop();
    }
    pop();

    int saved_fp = call_stack[SP];
    pop();
    FP = saved_fp;
}

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

/*
    현재 call_stack 전체를 출력합니다.
    해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
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");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    if (FP == SP) {
        push(-1, "Return Address");
    }
    else {
        push(-1, "Return Address");
        push(-1, "func1 SFP");
        FP = SP;
    }
    printf("func1 start \n");
    push(arg1, "arg1");
    push(arg2, "arg2");
    push(arg3, "arg3");
    int var_1 = 100;
    push(var_1, "var_1");
```

```
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func2(int arg1, int arg2)
{
    printf("func2 start \n");
    int var_2 = 200;
    push(var_2, "var_2");

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();

    int input_val;
    printf("func3에 넘길 정수 입력: ");
    scanf("%d", &input_val);
    func3(input_val);

    print_stack();

    pop();
}



void func3(int arg1)
{
        if (SP + 2 >= STACK_SIZE) {
        printf("ERROR: Stack overflow 위험으로 func3 실행 중단\n");
        return;
    }
    printf("func3 start \n");
    int var_3 = 300;
```

```
    int var_4 = 400;
    push(var_3, "var_3");
    push(var_4, "var_4");

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();

    pop();
    pop();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    printf("main start \n");
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
    pop();
    return 0;
}
```

하지만 이와같이 코드를 작성하면 scanf함수에서 stack overflow가 발생할 수 있다. 이를 막기 위해 SP가 가득찬다면 호출을 막는 코드를 추가하였다.