Callstack

먼저 아래의 코드를 보고 제일 먼저 든 생각은 함수가 stack에 쌓이는 과정을 line by line으로 구현해 보는 것이었습니다.

```
/*  call_stack

   실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
   원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

   ==========================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 배열

/*  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]` → SP = 1, ...

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

```
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)
{
   int var_1 = 100;

   // 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)
{
    int var_2 = 200;

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


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

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


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

우선 push 과정만 line by line으로 짜보았습니다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

    ============================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>
#include <string.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 배열

/*  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]` → SP = 1, ...

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

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)
{
    int var_1 = 100;

    int args[]={arg1, arg2, arg3};
    char args_name[][10]={"arg1", "arg2", "arg3"};

    for (int j=2; -1 < j; j--)
    {
        call_stack[SP] = args[j];
        strcpy(stack_info[SP], args_name[j]);
        SP++;
    }
    call_stack[SP] = -1;
    strcpy(stack_info[SP], "Return address");
    SP++;
    call_stack[SP] = -1;
    strcpy(stack_info[SP],"func1 SFP");
    FP = SP;
    SP++;
    call_stack[SP] = var_1;
    strcpy(stack_info[SP],"var_1");
    SP++;
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}
```

```
void func2(int arg1, int arg2)
{
   int var_2 = 200;

   int args_2[] = {arg1, arg2};
   char args_2_name[][10] = {"arg1", "arg2"};

   for (int j=1; -1 < j; j--)
   {
      call_stack[SP] = args_2[j];
      strcpy(stack_info[SP], args_2_name[j]);
      SP++;
   }
   call_stack[SP] = -1;
   strcpy(stack_info[SP],"Return address");
   SP++;
   call_stack[SP] = -1;
   strcpy(stack_info[SP], "func2 SFP");
   FP = SP;
   SP++;
   call_stack[SP] = var_2
   strcpy(stack_info[SP], "var_2");
   SP++;
   print_stack();
   func3(77);
   // func3의 스택 프레임 제거 (함수 에필로그 + pop)
   print_stack();
}


void func3(int arg1)
{
   int var_3 = 300;
   int var_4 = 400;

   int vars[] = {var_3, var_4};
   char vars_name[][10] = {"var_3", "var_4"};
   call_stack[SP] = arg1;
   strcpy(stack_info[SP], "arg1");
   SP++;
   call_stack[SP] = -1;
   strcpy(stack_info[SP], "Return address");
   SP++;
   call_stack[SP] = -1;
   strcpy(stack_info[SP], "func3 SFP");
   FP = SP;
   SP++;
```

```
    for (int k=1; -1 < k; k--)
    {
        call_stack[SP] = vars[k];
        strcpy(stack_info[SP], vars_name[k]);
        SP++;
    }
    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
}


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

각 함수마다 일일이 매개변수, Return address, 함수의 SFP, 지역변수를 저장하다보니 코드가 너무 길고 지저분해졌습니다. 지역변수나 매개변수를 개수별로 for 문을 돌려보았지만 그나마도 코드가 너무 일차원적이고 많은 줄을 차지하여 각 함수의 매개변수, 지역변수 를 포인터배열로 받고 매개변수의 개수, 지역변수의 개수, 함수의 이름을 매개변수로 받아 stack에 쌓는 push라는 함수를 구현해보기로 하였습니다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

    ============================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>
#include <string.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 배열

/*  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]` → SP = 1, ...

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

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");
}

void push (int *args, char **arg_names, int arg_size, int *locals, char **local_names, int local_si
ze, const char *func_name)
{
    for (int i = arg_size -1 ; i >= 0 ; i--)
    {
        call_stack[SP] = args[i];
        strcpy(stack_info[SP], arg_names[i]);
        SP++;
    }

    call_stack[SP] = -1;
    strcpy(stack_info[SP], "Return address");
    SP++;

    call_stack[SP] = -1;
    sprintf(stack_info[SP], "%s SFP", func_name);
    FP = SP;
    SP++;

    for (int j = 0; j < local_size ; j++)
    {
        call_stack[SP] = locals[j];
        strcpy(stack_info[SP], local_names[j]);
        SP++;
    }


}

//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    int args[] = {arg1, arg2, arg3};
    char *arg_names[] = {"arg1", "arg2", "arg3"};

    int locals[] = {var_1};
    char *local_names[] = {"var_1"};
```

```
    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func1");

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


void func2(int arg1, int arg2)
{
    int var_2 = 200;

    int args[] = {arg1, arg2};
    char *args_name[] = {"arg1", "arg2"};

    int locals[] = {var_2};
    char *local_names[] = {"var_2"};

    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func2");

    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    print_stack();
}


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    int args[] = {arg1};
    char *args_name[] = {"arg1"};

    int locals[] = {var_3, var_4};
    char *locals_names[] = {"var_3", "var_4"};

    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func3");

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

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

push 함수를 구현하다보니 func이 중첩됨에 따라 각 함수의 FP를 저장할 필요가 있어보여 FP_stack 배열을 전역변수로 선언하고 FP를 관리할 수 있도록 했습니다. 그리고 pop 함수는 간단하게 지역변수의 개수와 매개변수의 개수에 2(Return address와 함수의 SFP만큼)를 더한 만큼 call_stack을 비우고 stack_info를 초기화하는 작업을 해주었습니다. pop 함수를 실행할때 이전 함수의 FP로 제대로 이동할 수 있도록 구현해봤습니다.

```
/*  call_stack

   실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
   원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

   ==========================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>
#include <string.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 배열
```

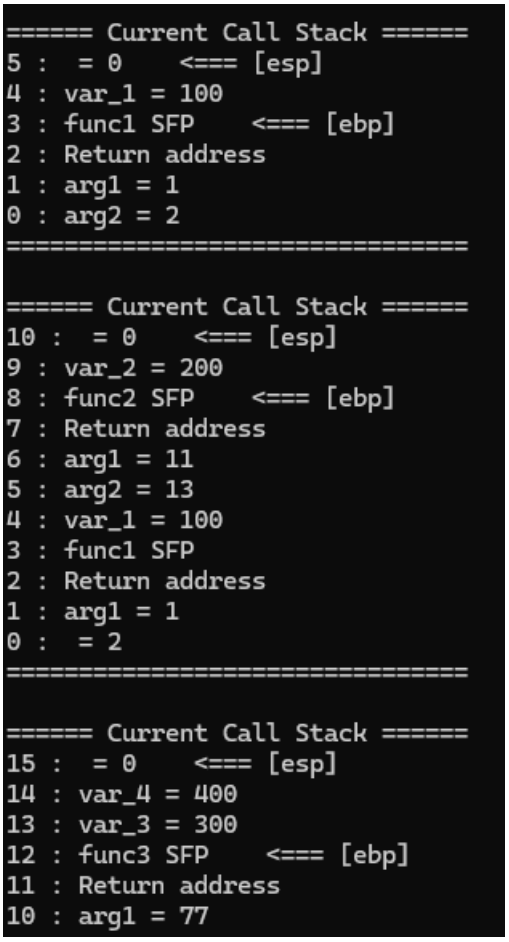
```
/*  SP (Stack Pointer), FP (Frame Pointer)

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

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;
int FP_stack[3]; //각 함수의 FP를 저장하는 배열

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");
}
```

```
//매개변수 배열, 매개변수 이름 배열, 매개변수 개수, 지역변수 배열, 지역변수 이름 배열, 지역변수 개수, 함수 이름을 인자로 받아 stack에 쌓는 함수
void push (int *args, char **arg_names, int arg_size, int *locals, char **local_names, int local_size, const char *func_name)
{
   for (int i = arg_size -1 ; i >= 0 ; i--)
   {
      call_stack[SP] = args[i];
      strcpy(stack_info[SP], arg_names[i]);
      SP++;
   }

   call_stack[SP] = -1;
   strcpy(stack_info[SP], "Return address");
   SP++;

   call_stack[SP] = -1;
   sprintf(stack_info[SP], "%s SFP", func_name);
   FP = SP;
   SP++;

   for (int j = 0; j < local_size ; j++)
   {
      call_stack[SP] = locals[j];
      strcpy(stack_info[SP], local_names[j]);
      SP++;
   }


}

//매개변수 개수와 지역변수 개수를 받아 그것에 2를 더한 만큼을 stack에서 비우고 초기화하는 함수
void pop (int arg_size, int local_size)
{
   int total_pop = arg_size + local_size + 2;

   for (int k = 0 ; k < total_pop ; k++)
   [
      SP--;
      call_stack[SP] = -1;
      strcpy(stack_info[SP],"");
   ]

   FP = FP_stack[SP]
}

//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
```

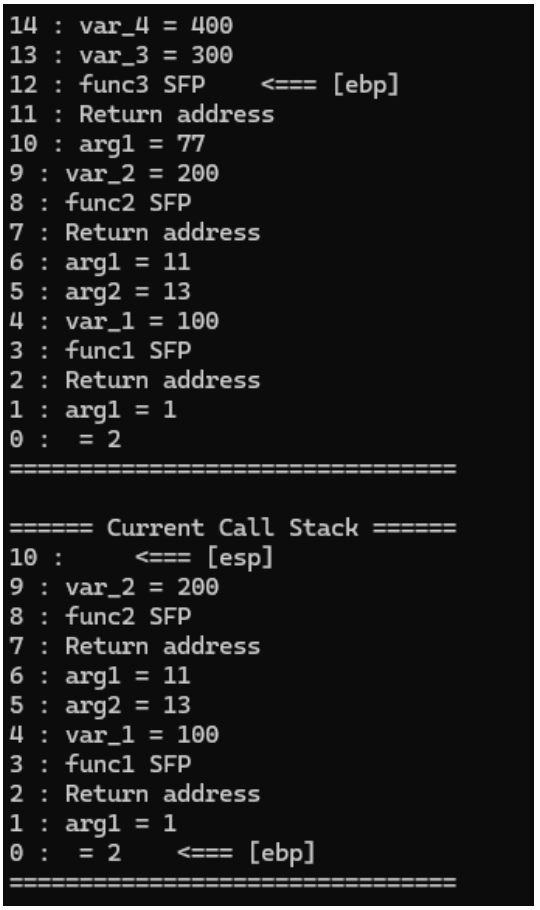
```
{
    int var_1 = 100;

    int args[] = {arg1, arg2, arg3};
    char *arg_names[] = {"arg1", "arg2", "arg3"};

    int locals[] = {var_1};
    char *local_names[] = {"var_1"};

    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func1");

    print_stack();
    func2(11, 13);

    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    pop (sizeof(args)/sizeof(int), sizeof(locals)/sizeof(int));

    print_stack();
}


void func2(int arg1, int arg2)
{
    int var_2 = 200;

    int args[] = {arg1, arg2};
    char *args_name[] = {"arg1", "arg2"};

    int locals[] = {var_2};
    char *local_names[] = {"var_2"};

    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func2");

    print_stack();
    func3(77);

    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    pop (sizeof(args)/sizeof(int), sizeof(locals)/sizeof(int));

    print_stack();
}


void func3(int arg1)
{
    int var_3 = 300;
```

```
    int var_4 = 400;

    int args[] = {arg1};
    char *args_name[] = {"arg1"};

    int locals[] = {var_3, var_4};
    char *locals_names[] = {"var_3", "var_4"};

    push(args, arg_names, sizeof(args)/sizeof(int), locals, local_names, sizeof(locals)/sizeof(int),
"func3");

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

    print_stack();
}


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

완성된 코드가 결과 예시처럼 잘 작동되는지 컴파일해보고 오류가 발생하면 수정해주었습니다. 우선 자잘한 문법 실수가 있어 수정해주었고 push 에서 FP_stack에 이전 FP를 저장하고 FP를 새로운 SP로 업데이트 하는 과정이 누락되어 추가해주었습니다. 실행해보았더니 push 함수로의 매개변수 전달은 잘 되고있으나 실행 결과가 정확하지 않아 다시 코드를 검토해보았습니다.



어딘가 시원찮은 결과들.....



단단히 오류가 있는 모습.

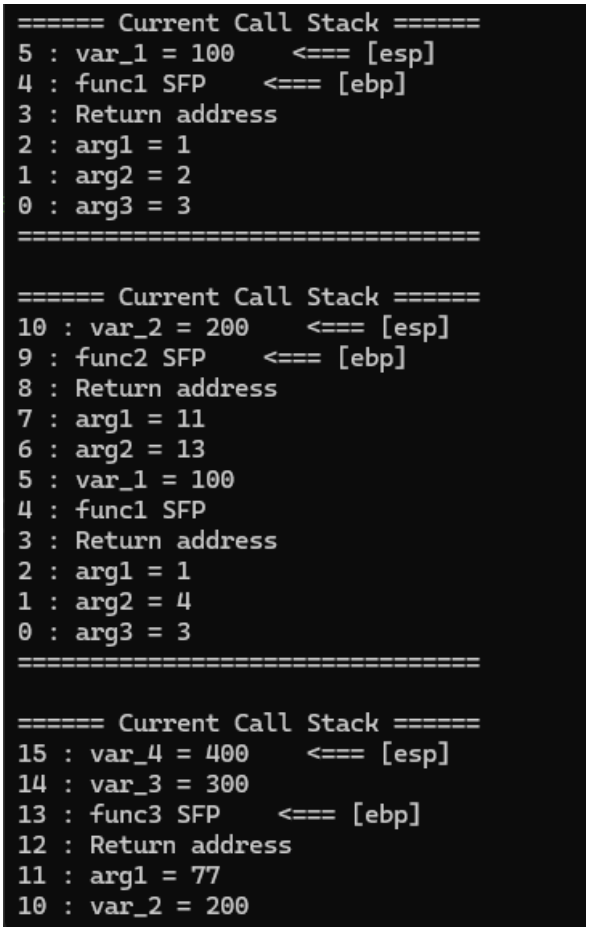
다행히 오류는 어렵지 않게 찾아낼 수 있었습니다. SP가 -1 이라는 것은 스택이 비워져있음을 의미하는 것이므로 0부터 값이 저장되어야합니다. 따라서 SP에 변수나 주소를 집어넣고 SP를 증가시키는 것이 아니라 SP++로 먼저 SP를 1 증가시킨 후에 값을 집어넣어야 했습니다. pop 함수 역시 현재의 SP를 비워주고 SP—를 통해 1 감소시키는 순서로 진행했습니다.

```
//매개변수 배열, 매개변수 이름 배열, 매개변수 개수, 지역변수 배열, 지역변수 이름 배열, 지역변수 개수, 함수 이름을 인자로 받아 stack에 쌓는 함수
void push (int *args, char **arg_names, int arg_size, int *locals, char **local_names, int local_size, const char *func_name)
{
   for (int i = arg_size -1 ; i >= 0 ; i--)
   {
      call_stack[SP] = args[i];
      strcpy(stack_info[SP], arg_names[i]);
      SP++;
   }

   call_stack[SP] = -1;
   strcpy(stack_info[SP], "Return address");
   SP++;
```

```
    call_stack[SP] = -1;
    sprintf(stack_info[SP], "%s SFP", func_name);
    FP = SP;
    SP++;

    for (int j = 0; j < local_size ; j++)
    {
        call_stack[SP] = locals[j];
        strcpy(stack_info[SP], local_names[j]);
        SP++;
    }


}

//매개변수 개수와 지역변수 개수를 받아 그것에 2를 더한 만큼을 stack에서 비우고 초기화하는 함수
void pop (int arg_size, int local_size)
{
    int total_pop = arg_size + local_size + 2;

    for (int k = 0 ; k < total_pop ; k++)
    [
        SP--;
        call_stack[SP] = -1;
        strcpy(stack_info[SP],"");
    ]

    FP = FP_stack[SP]
}
```

```
//매개변수 배열, 매개변수 이름 배열, 매개변수 개수, 지역변수 배열, 지역변수 이름 배열, 지역변수 개수, 함수 이름을 인자로 받아 stack에 쌓는 함수
void push(int* args, char** arg_names, int arg_size, int* locals, char** local_names, int local_size, const char* func_name)
{
    for (int i = arg_size - 1; i >= 0; i--)
    {
        SP++;
        call_stack[SP] = args[i];
        strcpy(stack_info[SP], arg_names[i]);
    }

    SP++;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], "Return address");
```

```
    SP++;
    call_stack[SP] = -1;
    sprintf(stack_info[SP], "%s SFP", func_name);
    FP_stack[SP] = FP;
    FP = SP;

    for (int j = 0; j < local_size; j++)
    {
        SP++;
        call_stack[SP] = locals[j];
        strcpy(stack_info[SP], local_names[j]);
    }


}

//매개변수 개수와 지역변수 개수를 받아 그것에 2를 더한 만큼을 stack에서 비우고 초기화하는 함수
void pop(int arg_size, int local_size)
{
    int total_pop = arg_size + local_size + 2;

    for (int k = 0; k < total_pop; k++)
    {
        call_stack[SP] = -1;
        strcpy(stack_info[SP], "");
        SP--;
    }

    FP = FP_stack[SP];
}
```



올바르게 스택에 쌓인 모습

위와 같이 코드가 마무리 되는 줄 알았으나....

```
====== Current Call Stack ======
5 : var_1 = 100    <=== [esp]
4 : func1 SFP    <=== [ebp]
3 : Return address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : func2 SFP    <=== [ebp]
8 : Return address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return address
2 : arg1 = 1
1 : arg2 = 4
0 : arg3 = 3
================================

====== Current Call Stack ======
15 : var_4 = 400    <=== [esp]
14 : var_3 = 300
13 : func3 SFP    <=== [ebp]
12 : Return address
11 : arg1 = 77
10 : var_2 = 200
```

func1dml arg2값 이상

```
5 : var_1 = 9
4 : func1 SFP
3 : Return address
2 : arg1 = 1
1 : arg2 = 4
0 : arg3 = 3
================================

====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : func2 SFP
8 : Return address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 9
4 : func1 SFP
3 : Return address
2 : arg1 = 1
1 : arg2 = 4    <=== [ebp]
0 : arg3 = 3
================================

====== Current Call Stack ======
4 : func1 SFP    <=== [esp]
3 : Return address
2 : arg1 = 1
1 : arg2 = 4
0 : arg3 = 3
================================
```

ebp 값에 문제 있는 것으로 추정

멀쩡하던 arg2의 값이 갑자기 자기 맘대로 변경되는 현상이 발견됩니다. 아래까지 결과를 살펴보니 ebp 값에도 문제가 있는 것으로 추정됩니다.

원인은 크게 두 가지로 살펴볼 수 있습니다. 먼저, FP_stack 배열의 크기를 함수의 크기와 유사하게 정해놓고 SP를 인자로 사용한 점입니다. SP를 인자로 활용하여 FP_stack을 표현하기 위해 FP_stack의 크를 STACK_SIZE로 설정하였습니다. 초기의 배열 값은 전부 0으로 저장하고 FP로 지정되는 번째의 배열에 FP 값을 업데이트 하도록 했습니다. pop함수에서는 FP_stack에 저장된 값이 0이 아닐때만 해당 배열 안에 저장된 값이 다시 FP가 되고(이전 FP로 복원) 해당 값은 0으로 초기화(스택 해제 후 FP값도 반환)되도록 if문을 추가해주었습니다. 두번째로는 pop 함수로 전달되는 매개변수의 문제입니다. 한 함수를 해제하기 위한 pop 함수는 해당 함수 밖에 들어있기 때문에 (ex, func3을 해제하기 위한 pop 함수는 func2에 들어있음) 해제해야하는 arg와 local의 크기를 pop이 사용된 함수 내에서의 변수들로는 제대로 표현할 수 없다는 것입니다. 어쩔 수 없이 이 부분은 일단 수기로 숫자를 세어 집어넣어줬습니다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

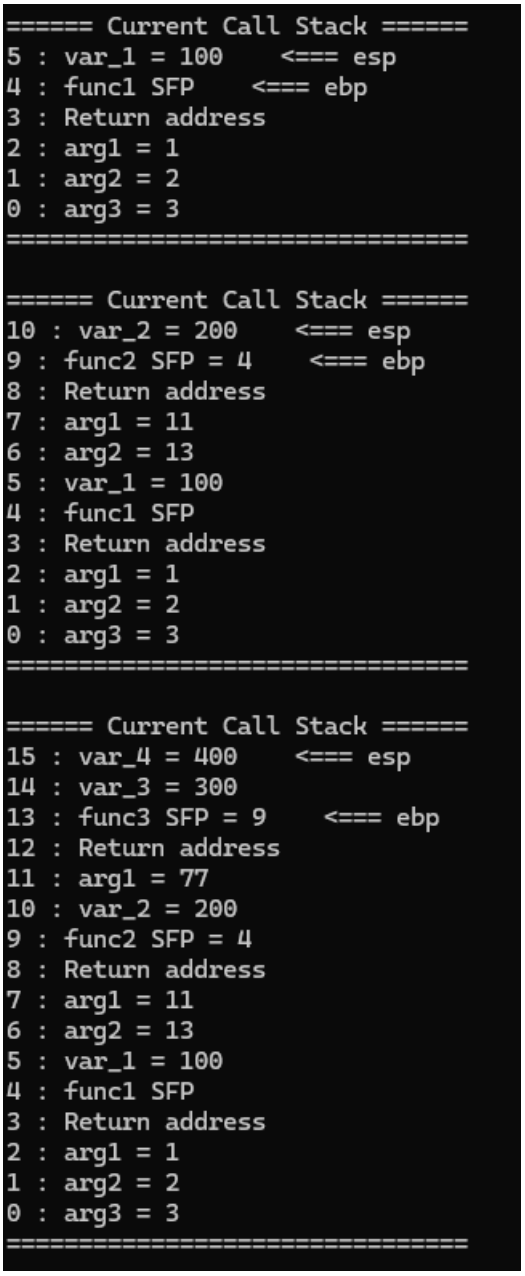
```
   ============================call_stack 저장 규칙============================
   매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
   Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
   반환 주소값을 push할 경우       : -1
   ============================================================================


   ============================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"
   ============================================================================
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.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 배열

/*  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]` → SP = 1, ...

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;
int FP_stack[STACK_SIZE]; //각 함수의 FP를 저장하는 배열, 0으로 초기화

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 if (FP_stack[i] != -1 && FP_stack[i]!=0)
            printf("%d : %s = %d", i, stack_info[i], FP_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");
}

//매개변수 배열, 매개변수 이름 배열, 매개변수 개수, 지역변수 배열, 지역변수 이름 배열, 지역변수 개수, 함수 이
void push(int* args, char** arg_names, int arg_size, int* locals, char** local_names, int local_size,
{
    for (int i = arg_size - 1; i >= 0; i--)
    {
        SP++;
        call_stack[SP] = args[i];
        strcpy(stack_info[SP], arg_names[i]);
    }

    SP++;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], "Return address");

    SP++;
    call_stack[SP] = -1;
    sprintf(stack_info[SP], "%s SFP", func_name);
    FP_stack[SP] = FP;
    FP = SP;

    for (int j = 0; j < local_size; j++)
    {
        SP++;
        call_stack[SP] = locals[j];
        strcpy(stack_info[SP], local_names[j]);
    }
```

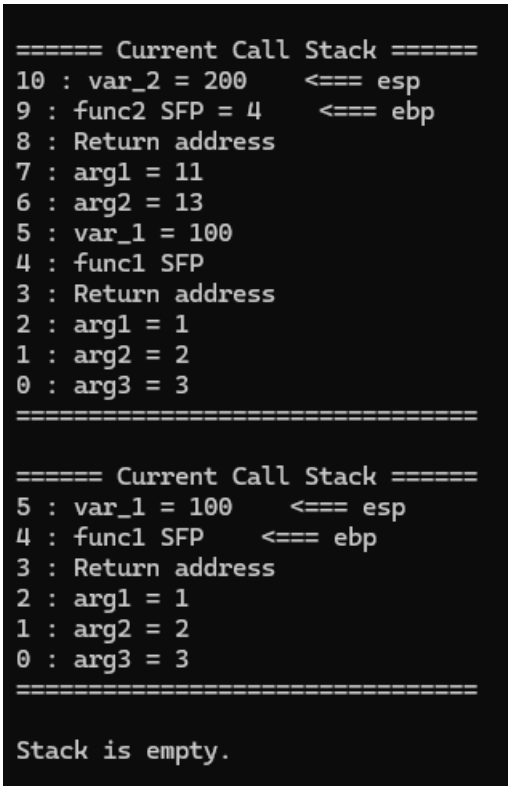
```
}

//매개변수 개수와 지역변수 개수를 받아 그것에 2를 더한 만큼을 stack에서 비우고 초기화하는 함수
void pop(int arg_size, int local_size)
{
   int total_pop = arg_size + local_size + 2;

   for (int k = 0; k < total_pop; k++)
   {
      call_stack[SP] = -1;
      strcpy(stack_info[SP], "");
      SP--;

      if (FP_stack[SP] != 0)
      {
         FP = FP_stack[SP];
         FP_stack[SP] = 0;
      }
   }
}

//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
   int var_1 = 100;

   int args[] = { arg1, arg2, arg3 };
   char* arg_names[] = { "arg1", "arg2", "arg3" };

   int locals[] = { var_1 };
   char* local_names[] = { "var_1" };

   push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(int),

   print_stack();
   func2(11, 13);

   // func2의 스택 프레임 제거 (함수 에필로그 + pop)
   pop(2, 1);

   print_stack();
}


void func2(int arg1, int arg2)
{
```

```
    int var_2 = 200;

    int args[] = { arg1, arg2 };
    char* arg_names[] = { "arg1", "arg2" };

    int locals[] = { var_2 };
    char* local_names[] = { "var_2" };

    push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(int),

    print_stack();
    func3(77);

    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    pop(1, 2);

    print_stack();
}


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    int args[] = { arg1 };
    char* arg_names[] = { "arg1" };

    int locals[] = { var_3, var_4 };
    char* local_names[] = { "var_3", "var_4" };

    push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(int),

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

    print_stack();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);

    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    pop(3,1);
    print_stack();
```

```
    return 0;
}
```



결과 화면_1



결과 화면_2

코드를 완성하고 보니 call_stack 저장 규칙을 따르지 않은 부분을 발견했습니다. 이때까지 SFP도 Return address와 마찬가지로 call_stack에 -1을 저장하고 있었습니다. 올바른 규칙에 따르면 SFP를 push할 경우 call_stack에는 call_stack에서의 index를 저장하면 됩니다. 이렇게 하면 사실 FP_stack 배열은 사용할 필요도 없었던거죠.

따라서 FP_stack 배열을 삭제해주고 call_stack에 SFP의 index가 그대로 쌓이도록 하여 FP를 관리해주었습니다.

```
/*  call_stack

   실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 메모리를 구현합니다.
   원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에서는 -1로 대체합니다.

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

   ==========================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"
   ========================================================================
======
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.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 배열

/*  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]` → SP = 1, ...

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

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");
}

//매개변수 배열, 매개변수 이름 배열, 매개변수 개수, 지역변수 배열, 지역변수 이름 배열, 지역변수 개수, 함수 이름을 인자로 받아 stack에 쌓는 함수
void push(int* args, char** arg_names, int arg_size, int* locals, char** local_names, int local_size, const char* func_name)
{
   //매개변수 push(역순)
   for (int i = arg_size - 1; i >= 0; i--)
   {
      SP++;
      call_stack[SP] = args[i];
      strcpy(stack_info[SP], arg_names[i]);
   }

   //Return address push
   SP++;
```

```
    call_stack[SP] = -1;
    strcpy(stack_info[SP], "Return address");

    //SFP push (call_stack에서의 index가 그대로 저장)
    SP++;
    call_stack[SP] = FP;
    sprintf(stack_info[SP], "%s SFP", func_name);
    FP = SP;

    //지역변수 push(순차)
    for (int j = 0; j < local_size; j++)
    {
        SP++;
        call_stack[SP] = locals[j];
        strcpy(stack_info[SP], local_names[j]);
    }
}

//매개변수 개수와 지역변수 개수를 받아 지역변수의 개수만큼 초기화 한 다음, FP를 이전 것으로 복원한 다음,
다시 매개변수+2(SFP&Return address)만큼 초기화
void pop(int arg_size, int local_size)
{
    //지역변수 pop
    for (int k = 0; k < local_size; k++)
    {
        call_stack[SP] = -1;
        strcpy(stack_info[SP], "");
        SP--;
    }

    //SFP를 이전의 것으로 복원
    FP = call_stack[SP];

    //SFP + Return address + 매개변수 pop
    for (int r = 0; r < arg_size+2; r++)
    {
        call_stack[SP] = -1;
        strcpy(stack_info[SP], "");
        SP--;
    }
}

//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    int args[] = { arg1, arg2, arg3 };
```

```
    char* arg_names[] = { "arg1", "arg2", "arg3" };

    int locals[] = { var_1 };
    char* local_names[] = { "var_1" };

    push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(i
nt), "func1");

    print_stack();
    func2(11, 13);

    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    pop(2, 1);

    print_stack();
}


void func2(int arg1, int arg2)
{
    int var_2 = 200;

    int args[] = { arg1, arg2 };
    char* arg_names[] = { "arg1", "arg2" };

    int locals[] = { var_2 };
    char* local_names[] = { "var_2" };

    push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(i
nt), "func2");

    print_stack();
    func3(77);

    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    pop(1, 2);

    print_stack();
}


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    int args[] = { arg1 };
    char* arg_names[] = { "arg1" };
```

```
    int locals[] = { var_3, var_4 };
    char* local_names[] = { "var_3", "var_4" };

    push(args, arg_names, sizeof(args) / sizeof(int), locals, local_names, sizeof(locals) / sizeof(i
nt), "func3");

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

    print_stack();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);

    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    pop(3,1);
    print_stack();
    return 0;
}
```

위와 같이 완성된 코드로 각 함수가 호출될 때 스택에 어떻게 쌓이는지 구현해볼 수 있었습니다. pop 함수로 전달할 매개변수도 단순히 숫자를 계산해서 전달하는 것이 아니라 다른 변수들을 사용하고 싶었지만 마땅한 방법이 떠오르지 않았습니다. 매개변수의 개수는 변수로 전달할 수 있어도 함수 내부에 있는 지역변수는 해당 함수를 빠져나오면 도저히 접근할 수 없고 이를 표현하는 것도 불가능할 것 같아 개수를 세어 대입하는 것으로 마무리하였습니다.