Cykor 과제 보고서(1주차)

일단 과제를 하기에 앞서 base.c 코드를 읽어보면서 과제 방향성에 대해 잡으려고 노력했습니다. 조금씩 나누어서 세세하게 보도록 하겠습니다.

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 대
    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"
    ============================================================
*/
```

일단 주석에 쳐 져있는 설명대로, 실제 시스템이 메모리에 쌓이는 것을 배열로 구현하는 것이 이번 과제의 목표입니다. 일단 스택에 쌓이는 값들은 매개변수, 지역변수, 저장된 프레임 포인터, 리턴 주소 4가지가 있고 리턴 주소, SFP의 경우 실제는 가상 메모리 주소가 들어가지만, 이 과제에서는 -1 또는 index로 대체합니다. 그리고 call_stack 규칙과 stack_info 규칙이 다음과 같이 있습니다.

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

call_stack은 실제 데이터가 저장되는 배열이며, 변수값, 리턴주소, 저장된 FP 등이 들어갑니다. stack_info는 call_stack의 인덱스 위치를 설명합니다. SP(Stack Pointer)는 현재 스택의 맨 위를 가리키며, 스택이 비었을 때는 -1입니다. 그리고 push하면 더해지고, pop하면 빼지는 값입니다. FP(Frame Pointer)는 현재 활성화된 함수의 스택 프레임 시작 위치를 나타냅니다. 새 함수를 호출할 경우 FP를 갱신하고, 종료하면 복구해야 합니다. 그리고 우리가 추가해야 할 func1, func2, func3가 선언되어 있습니다.

```
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");
}
```

과정을 보면

1. SP가 -1일 경우 “Stack is empty” 출력
2. for문을 통해 SP부터 0까지 내려가면서 출력
3. 값이 -1일 경우 stack_info[i](Return address)만 출력
4. -1이 아닐 경우 i, stack_info[i], call_stack[i] 출력
5. SP 위치에 [esp], FP 위치에 [ebp] 표시

이러한 형태입니다.

```
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();
}
```

func1, func2, func3의 경우 지역변수 값 선언하고, 스택 프레임 형성한 후 print_stack() 하고, 다른 함수 호출한 후 스택 프레임 제거하고, print_stack()하는 형태입니다.

그래서 일단 이번 과제를 진행하기 위해서는 스택이 쌓이고 없어지는 순서에 대해 정리해볼 필요가 있습니다. 코드를 토대로 다음과 같이 정리할 수 있습니다.

main(func1 호출) → func1 실행(스택 프레임 생성 및 print_stack()) → fuc2 호출(SP증가, FP 변경) → func2 실행(스택 프레임 생성 및 print_stack()) → func3 호출(SP 증가, FP 변경) → func3 실행(스택 프레임 생성 및 print_stack()) → func3 종료(스택 해제 + print_stack()) → func2 종료(스택 해제 + print_stack()) → func1 종료(스택 해제 + print_stack()) → main 종료 이런 상황입니다. 이제 이 상황에 맞게 stack_frame 형성 및 SP랑 FP 값을 조절해보러 가보겠습니다.

그래서 우리는 이러한 것들을 우선적으로 알 수 있었고, 이를 토대로 첫 번째 구현 코드를 다음과 같이 작성하였습니다.

```
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    call_stack[++SP] = arg1;
```

```
    strcpy(stack_info[SP], "arg1");

    call_stack[++SP] = arg2;
    strcpy(stack_info[SP], "arg2");

    call_stack[++SP] = arg3;
    strcpy(stack_info[SP], "arg3");

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

    call_stack[++SP] = FP;
    strcpy(stack_info[SP], "func1 SFP");

    FP = SP;

    call_stack[++SP] = var_1;
    strcpy(stack_info[SP], "var_1");
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    SP--;
    SP--;
    FP = call_stack[SP];
    SP--;
    SP--;
    SP--;
    SP--;
    print_stack();
}
```

func1은 다음과 같이 작성하였습니다. 이 코드에 대해 세세하게 설명하면, call_stack이 매개변수, 지역변수, SFP 등을 저장하는 곳 이기에, 매개변수가 들어왔을 때, Stack Pointer 값을 증가시켜주면서 call_stack에 저장을 합니다. 그 후 그 위에 Return Address 저장하고, 그 위에는 이전 Frame Pointer 저장해준 후, Frame pointer를 Stack pointer로 맞춰서 저장합니다. 이제 그 이후에 지역변수를 쌓기 시작합니다. 그 후 함수가 종료 될때는 지금

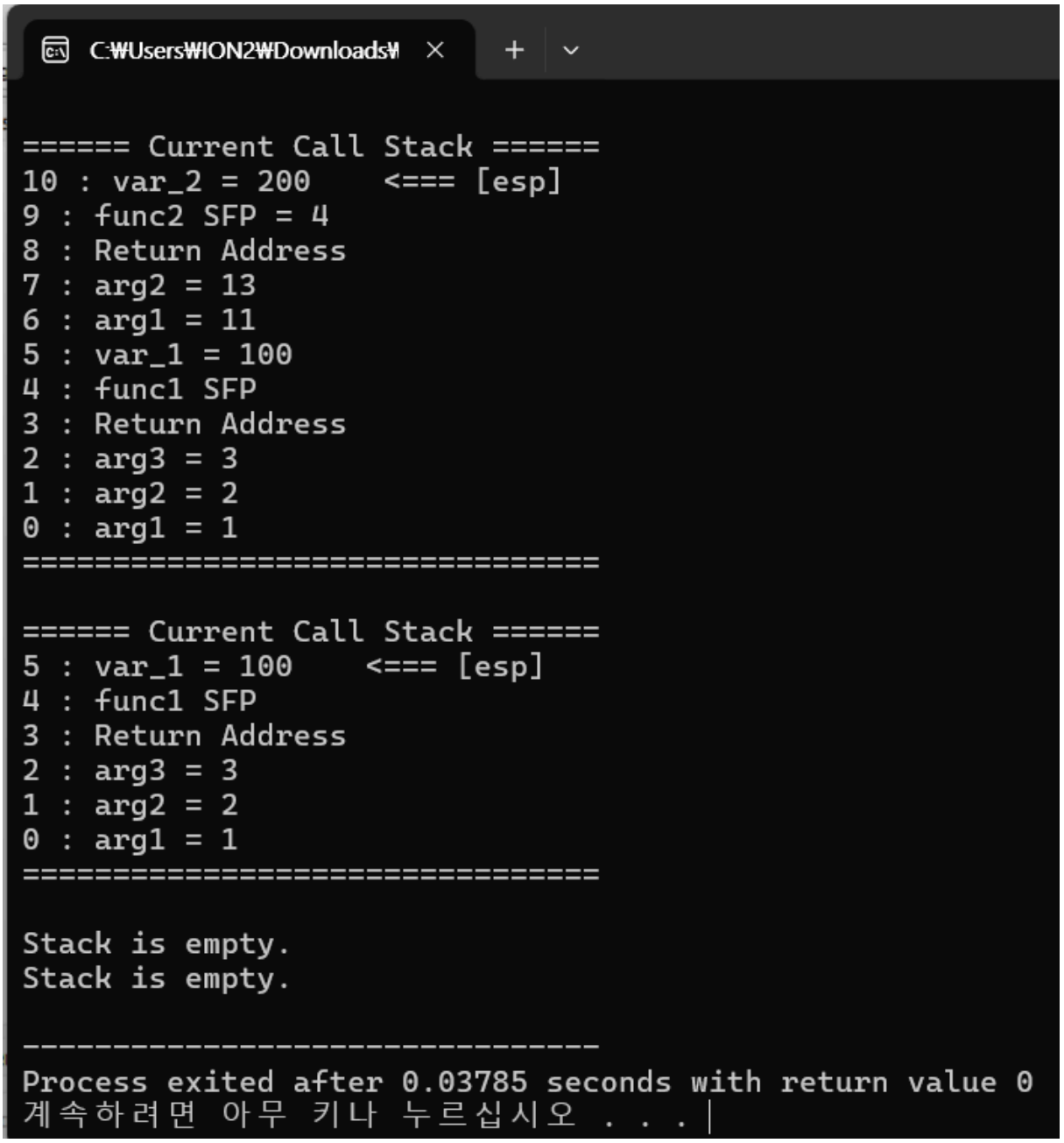
까지 넣었던 값들을 제거해주어야 하기에, SP - - 해서 var1 제거하고, 이전 FP 복원한 후, SFP제거하고, Return Address 제거한 후, arg1~arg3까지 3개 제거하기 위해 SP - - 를 3번 해줍니다.

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

   call_stack[++SP] = arg1;
   strcpy(stack_info[SP], "arg1");

   call_stack[++SP] = arg2;
   strcpy(stack_info[SP], "arg2");

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

   call_stack[++SP] = FP;
   strcpy(stack_info[SP], "func2 SFP");

   FP = SP;

   call_stack[++SP] = var_2;
   strcpy(stack_info[SP], "var_2");
   // func2의 스택 프레임 형성 (함수 프롤로그 + push)
   print_stack();
   func3(77);
   // func3의 스택 프레임 제거 (함수 에필로그 + pop)
   SP--;
   FP = call_stack[--SP];
   SP--;
   SP -= 2;
   print_stack();
}
```

func2는 다음과 같이 작성하였습니다. func2도 func1과 매우 유사하게 작성을 합니다. 다른 점이라면 매개변수 저장 수와 그에 따른 에필로그에서 SP - -의 수의 차이라고 볼 수 있습니다.

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

    call_stack[++SP] = arg1;
    strcpy(stack_info[SP], "arg1");

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

    call_stack[++SP] = FP;
    strcpy(stack_info[SP], "func3 SFP");

    FP = SP;

    call_stack[++SP] = var_3;
    strcpy(stack_info[SP], "var_3");

    call_stack[++SP] = var_4;
    strcpy(stack_info[SP], "var_4");
    // func3의 스택 프레임 형성 (함수 프롤로그 + push)

    SP--;
    SP--;
    FP = call_stack[--SP];
    SP--;
    SP--;
    print_stack();
}
```

func3는 다음과 같이 작성하였습니다. 여기도 마찬가지입니다. 매개변수와 지역변수 개수의 차이로 인한 SP - - 의 차이가 좀 있고, 다른 것은 동일합니다. 그리고 이 코드를 수행하였을 때, 다음과 같이 결과가 나왔습니다.



그런데 이렇게 보니 과제 결과 예시 코드랑 다르게 맞지 않고, 코드에 문제가 있어 다시 한 번 시도해보았습니다.

제가 다시 보니 주석을 잘 못 읽었더라고요. func1의 스택 프레임 제거는 main 함수에서 하는 것이었습니다. 그래서 코드를 다시 고쳤습니다.

```
/*  call_stack

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

   int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저
   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
#include <string.h>

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;
```

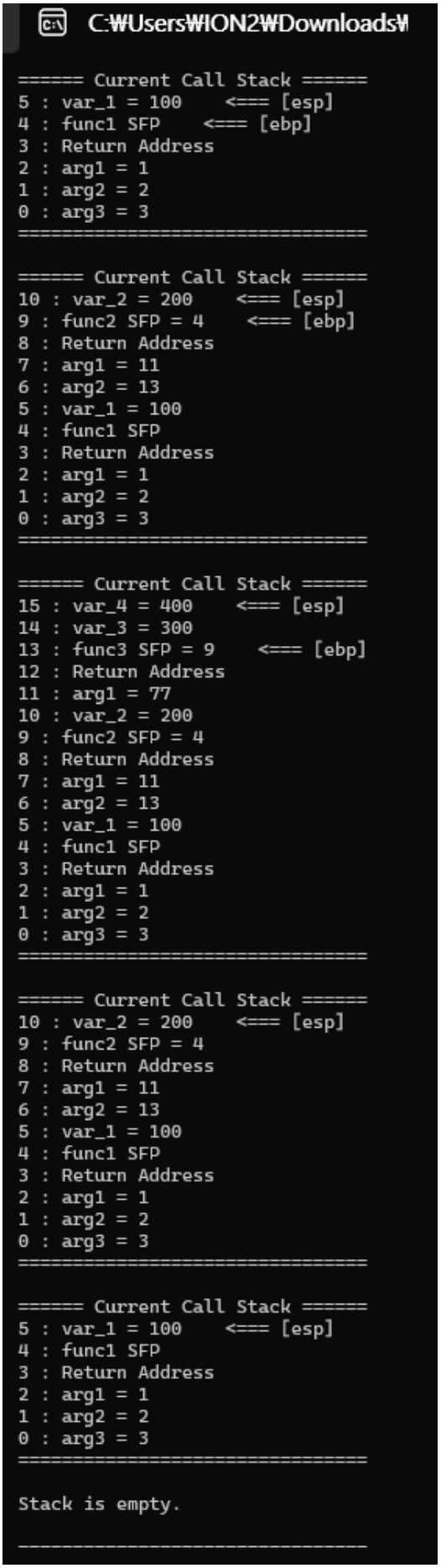
```
int FP = -1;

void 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;

   call_stack[++SP] = arg3;
   strcpy(stack_info[SP], "arg3");

   call_stack[++SP] = arg2;
   strcpy(stack_info[SP], "arg2");

   call_stack[++SP] = arg1;
   strcpy(stack_info[SP], "arg1");

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

   call_stack[++SP] = FP;
   strcpy(stack_info[SP], "func1 SFP");

   FP = SP;

   call_stack[++SP] = var_1;
   strcpy(stack_info[SP], "var_1");
   // func1의 스택 프레임 형성 (함수 프롤로그 + push)
   print_stack();
   func2(11, 13);
   // func2의 스택 프레임 제거 (함수 에필로그 + pop)
   SP--;
   FP = call_stack[--SP];
   SP--;
   SP -= 2;
   print_stack();
}


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

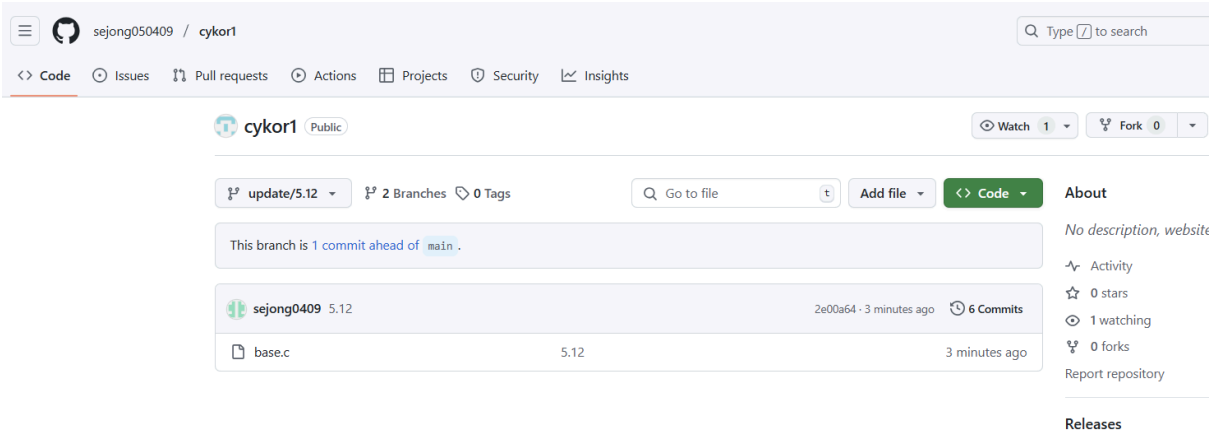
```
    int var_2 = 200;

    call_stack[++SP] = arg2;
    strcpy(stack_info[SP], "arg2");

    call_stack[++SP] = arg1;
    strcpy(stack_info[SP], "arg1");

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

    call_stack[++SP] = FP;
    strcpy(stack_info[SP], "func2 SFP");

    FP = SP;

    call_stack[++SP] = var_2;
    strcpy(stack_info[SP], "var_2");
    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    SP--;
    SP--;
    FP = call_stack[--SP];
    SP--;
    SP--;

    print_stack();
}


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

    call_stack[++SP] = arg1;
```

```
    strcpy(stack_info[SP], "arg1");

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

    call_stack[++SP] = FP;
    strcpy(stack_info[SP], "func3 SFP");

    FP = SP;

    call_stack[++SP] = var_3;
    strcpy(stack_info[SP], "var_3");

    call_stack[++SP] = var_4;
    strcpy(stack_info[SP], "var_4");
    // 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)
    SP--;
    SP--;
    FP = call_stack[SP];
    SP--;
    SP--;
    SP--;
    SP--;
    print_stack();
    return 0;
}
```

이렇게 코드를 다시 고쳤고요...(다음부턴 주석을 정확히 읽어야 겠다는 것을 깨닫는 과제였습니다...) 다른건 따로 바뀐게 없고, 이제 stack 매개변수 순서나, 지역변수 순서 및 스택 프레임 제거 함수 위치를 바꾸었습니다. 그래서 결과는 다음과 같이 나왔습니다.


C:₩Users₩ION2₩Downloads₩
====== 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 = 4 <=== [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 = 2
0 : arg3 = 3
==========
====== Current Call Stack ======
15 : var_4 = 400 <=== [esp]
14 : var_3 = 300
13 : func3 SFP = 9 <=== [ebp]
12 : Return Address
11 : arg1 = 77
10 : var_2 = 200
9 : func2 SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==========
====== Current Call Stack ======
10 : var_2 = 200 <=== [esp]
9 : func2 SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==========
====== Current Call Stack ======
5 : var_1 = 100 <=== [esp]
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==========
Stack is empty.

잘 나온 것을 확인할 수 있습니다. 그래서 고친 부분은 5.12 branch 만들어서 그 branch안에 commit 해서 집어넣었습니다. 즉 완성코드는 여기에 있습니다.



그리고 이 branch에 보고서도 넣으면서 이번 과제 마무리 하겠습니다! 감사합니다!