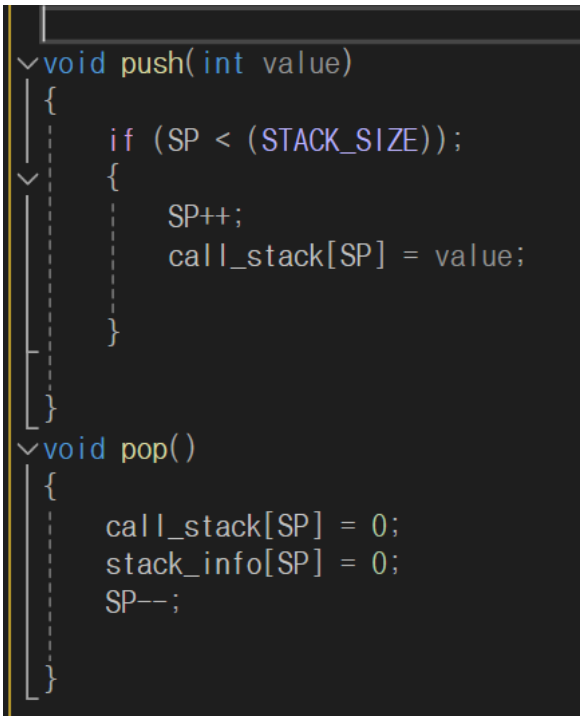
2025 Cykor 1week 과제

call stack 구현하기

1. pop, push 기능(함수) 구현

초기 상태



```
void push(int value)
{
    if (SP < (STACK_SIZE));
    {
        SP++;
        call_stack[SP] = value;

    }

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

}
```

간단히 변수를 입력받아서 SP값을 올린 후 그 SP값에 해당하는 인덱스에 변수에 저장되어 있는 값을 call_stack에 저장하는 방식이다. 하지만 생각해보면, 과제에서는 stack_info에서는 변수인 경우 변수의 이름을 저장해야 한다. 하지만 함수 프롤로그에서는 함수 인자를 stack에 저장하는데 그럼 나는 함수 인자의 개수를 알아야 하고 그 함수의 이름도 다른 변수에 저장해야 한다.

그래서 찾아본 것이 가변인자를 받는 매크로들이 저장되어 있는 헤더파일 stdarg.h이다.

va_list: 가변 인자 목록을 나타냄(정확히는 가변인자에 저장되어있는 값을 가르키는 포인터)

va_start(): 가변 인자 시작

va_arg(args, type): 가변인자 읽기

va_end(): 가변 인자 끝

정리해보면 va_list로 가변인자를 저장한 다음, va_start, va_arg를 통해 가변인자를 stack_info에 저장하면 될 것이다.

또한, sprintf함수를 통해 str문자열을 저장한다.

그럼 일단 프롤로그 부터 구현해보겠다.

프롤로그 함수 인자

FP에 담을 함수의 이름, 그 함수의 인자들과 지역변수들이 있겠다.

```
void Prologue(char* func_name, int count, ...);
{
    va_list args;
    va_start(args, count);
    for (int i = 0; i < count; i++) //함수 인자 stack에 push
    {
        int arg = va_arg(args, int);
        char temp[50];
        sprintf(temp, "arg%d", i + 1);
        push(arg, temp);



    }

}
```

이렇게 push에 함수 인자에 저장된 값과 인자 이름을 전달하여 각각 call_stack, stack_info에 저장하면 되므로 다시 push함수를 수정히면,

```
void push(int value, char value_name)
{
    if (SP < (STACK_SIZE));
    {
        SP++;
        call_stack[SP] = value;
        strcpy(stack_info[SP], value_name);
    }

}
```

strcpy로 stack_info에 저장하였다.

그다음, return address랑 SFP를 stack에 저장하는 것을 구현해보았다.

```
void Prologue(char* func_name, int count, ...);
{
    va_list args;
    va_start(args, count);
    for (int i = 0; i < count; i++) //함수 인자 stack에 push
    {
        int arg = va_arg(args, int);
        char temp[50];
        sprintf(temp, "arg%d", i + 1);
        push(arg, temp);



    }
    va_end(args);
    push(-1, "return address");//return address push

    char sfp_info[50];
    sprintf(sfp_info, " %s SFP", func_name); //SFP push


    push(FP, sfp_info);
    FP = SP;
```

return address와는 달리 sfp또한 함수의 이름을 받아와야 해서 sprintf함수를 사용했다.

그리고 마지막에는 FP를 SP로 갱신시켰다.

>> 처음에는 epilogue 함수 내부에서 갱신 시켰지만 출력값을 생각해보니 이전함수의 SFP가 저장된 index(SP)가 되어야 하므로 func함수 내에서 갱신 시켰다.

```
void func2(int arg1, int arg2)
{
    Prologue("func2", 2,arg1, arg2);
    FP = SP;//FP 갱신
    int FP2 = FP; //복원할 func1의 SFP 저장
    int* func2_SFP = &FP2;
    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)
    Epilogue()
    print_stack();
}


void func3(int arg1)
{
    Prologue("func3", 1, arg1);
    FP = SP;//FP 갱신
    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();
}
```

프롤로그를 통해서 지역변수와 return address, SFP를 push한 후 , FP를 갱신시키고 그 FP를 포인터로 저장하였다.

그다음은 에필로그 함수의 구현이다.

epilougue 과정에는 지역변수가 pop이 되어야한다. 그래서 일단 지역변수의 수, 매개변수의 수만큼 pop을 수행해야 하는 것을 알 수 있다. 그럼 에필로그 함수의 인자로 지역/매개

변수의 수를 가져오고 FP를 SFP를 통해 이전 함수로 복원하기 위해 fun내부의 선언한 포인터 변수를 매개 변수로 가져왔다.

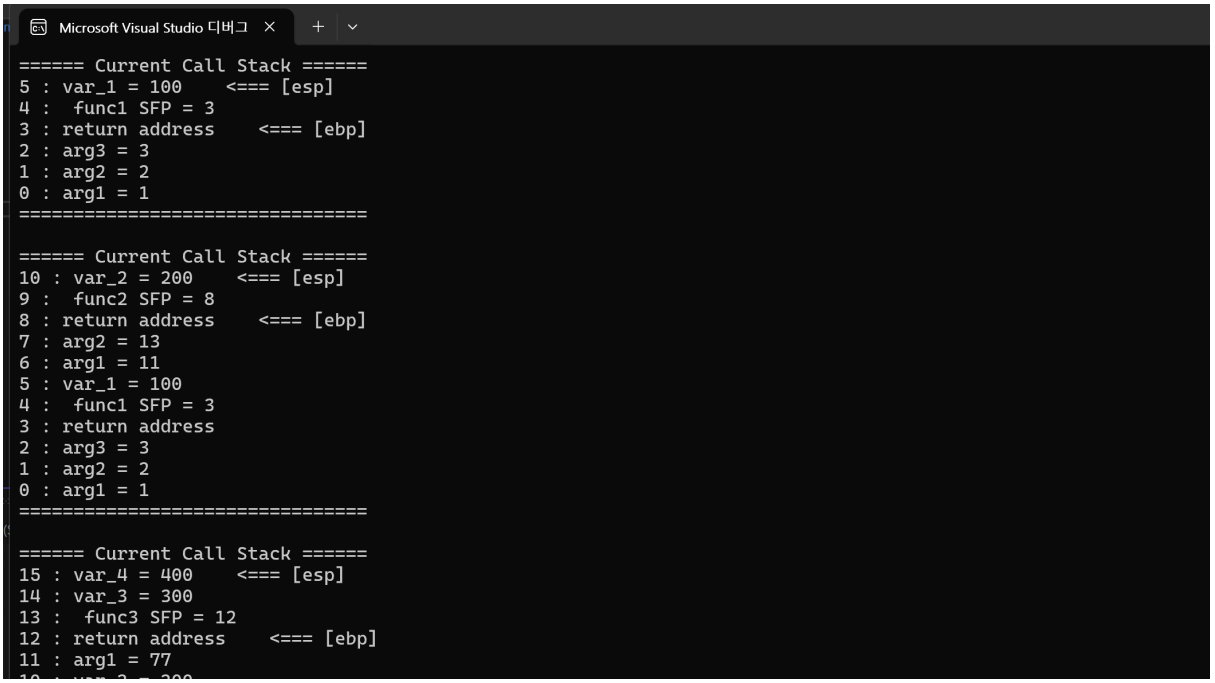
```
void Epilogue(int count_local, int count_arg, int *SFP)
{
    for (int i = 0; i < count_local; i++)//지역 변수 pop
    {
        pop();
    }

    FP = *SFP; //SFP 복원
    pop();
    pop();//return address 제거
    for (int i = 0; i < count_arg; i++)
    {
        pop();
    }
}
```

그리고 위에서 FP의 갱신을 fun함수 내부에서 한다고 했는데 그러면 프롤로그에서 FP push가 -1로 되어버리는 오류가 생겨서 그냥 FP 갱신을 에필로그에서 하도록 바꾸었다.

그리고

에필로그 함수까지 func함수에 넣어서 코드를 완성하였다.

그리고 디버깅 시 발생하는 오류를 수정한 후 실행해보니



지금 발생하는 문제점은

1. ebp가 return address를 가르키고 있다. → FP값이 잘못 되어있다.
2. 처음 current stack이 나올때 func SFP가 현재 함수의 FP의 index를 가르키고 있다.→ call_stack 값이 잘못 저장되어있다.
3. 매개변수 stack 순서가 거꾸로 되어있다.

1. 해결방안

FP를 1더해주면 된다. 지금 push함수에서는 SP만 1을 더하고 있어서 FP를 1 더하면 된다. 이때 value가 FP일때만 더해주면 되겠다.

```
void push(int value, char value_name[])
{
    if ((SP < (STACK_SIZE)-1)&&(!(value==FP)))
    {
        ++SP;
        call_stack[SP] = value;
        strcpy(stack_info[SP], value_name);
    }
    else
    {
        ++FP;
        ++SP;
        call_stack[SP] = value;
        strcpy(stack_info[SP], value_name);
    }

}
```

그리고 사진의 코드상에는 value가 FP가 아닐때만 SP를 더해준다고 하였는데

매우 비효율적이고 코드 에러가 나서 프롤로그 함수내에서

FP를 더하는 방식으로 진행하였다.

-수정 코드

```
void push(int value, char value_name[])
{
    if ((SP < (STACK_SIZE)-1))
    {
        ++SP;
        call_stack[SP] = value;
        strcpy(stack_info[SP], value_name);



    }

}
```

```
    char sfp_info[50];
    sprintf(sfp_info, " %s SFP", func_name); //SFP push
    FP = SP;
    ++FP;
    push(*bf_FP, sfp_info);
```

2. 해결방안

프롤로그 함수 내에서 지역변수로 이전 FP를 저장하고 그것을 포인터로 지정하여 참조하도록 하였다. 그러면 프롤로그 함수가 실행되면 1. 이전 SFP 저장→ SFP 포인터 갱신 → 다음 함수 프롤로그→ 갱신된 FP 저장→ ... 으로 계속 갱신 된다. 그리고 SFP push를 이전 FP를 call_stack에 담기게 하면 되겠다.

```
void Prologue(char* func_name, int count, ...)
{
    int bfFP = FP;

    int* bf_FP = &bfFP;//이전 FP
    va_list args;
    va_start(args, count);
    for (int i = count-1; i >=0; i--) //함수 인자 stack에 push
    {
        int arg = va_arg(args, int);
        char temp[50];
        sprintf(temp, "arg%d", i+1);
        push(arg, temp);



    }
    va_end(args);

    push(-1, "return address",bf_FP);//return address push


    char sfp_info[50];
```

3. 해결방안

va_list는 인자를 역순으로 불러올 수 없기에

```
Prologue("func1", 3,arg3, arg2, arg1);
```

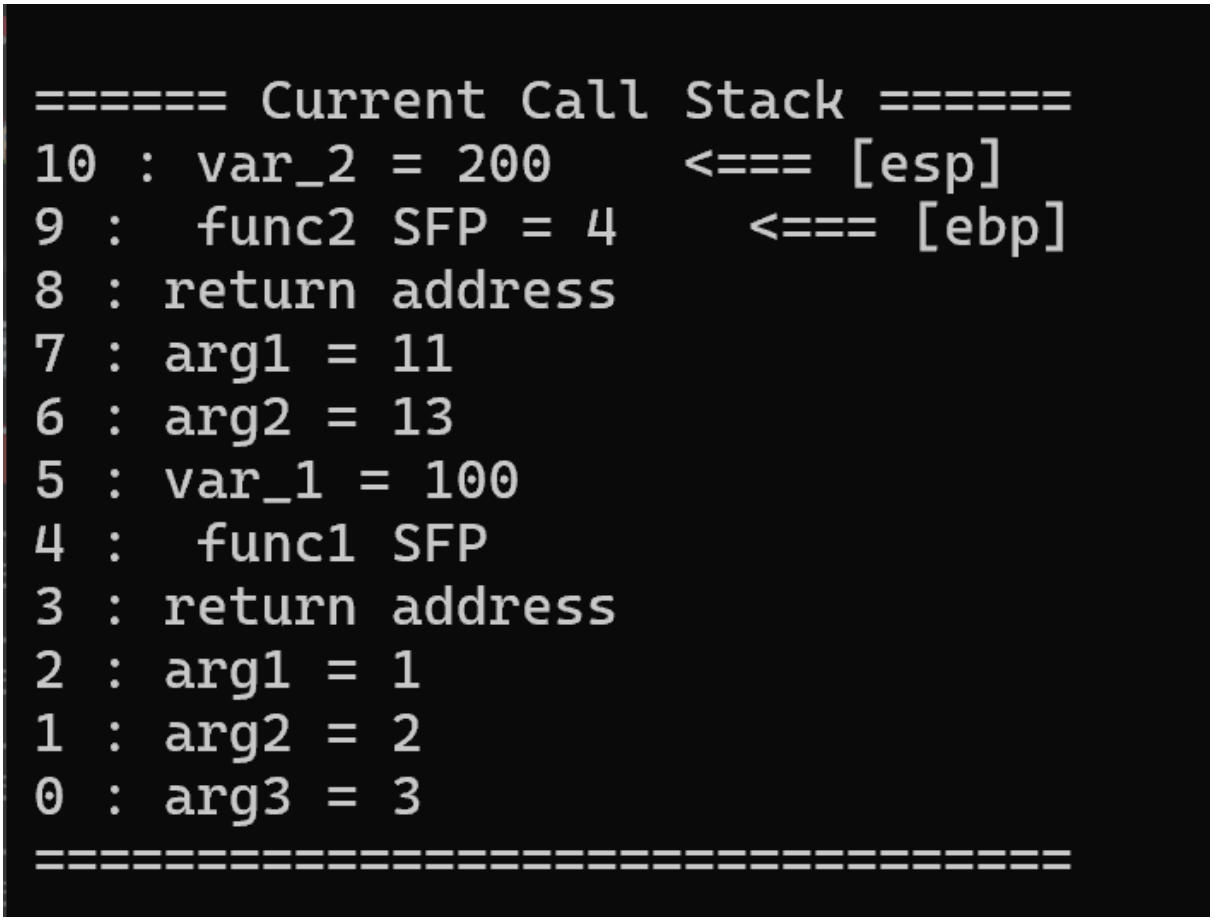
처럼 직접 매개변수의 순서를 바꿔준다.

이러한 오류들을 해결한 후 출력을 해보았다.

1. func1의 stack frame

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

2. func1+func2



SFP가 이전 함수 func1의 SFP의 i값을 가르키고 있다.

3. func1+func2+func3

4.함수 프롤로그

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

Stack is empty.
```

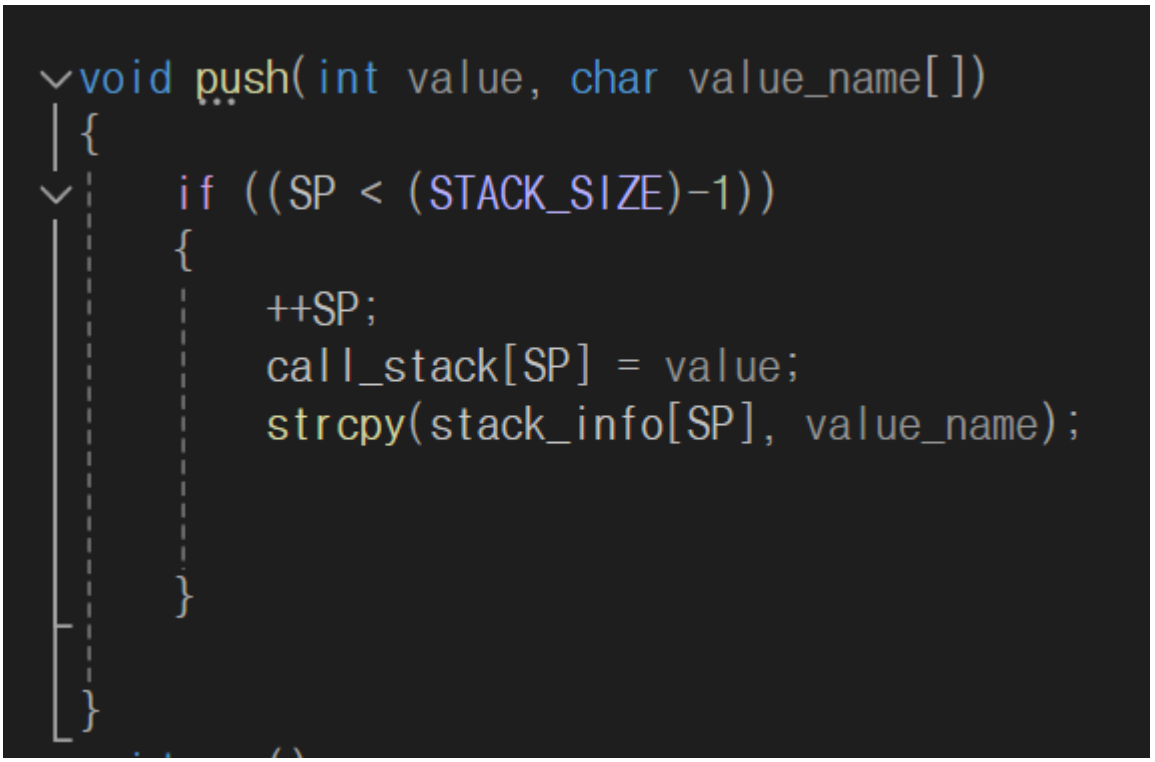
함수 프롤로그가 실행될 때 마다 이전 함수의 FP로 FP가 갱신되어 ebp가 그 SFP를 가르키는 모습이다.

지금 나의 코드는 func1,2,3 내부에서 그 함수의 FP를 저장후(FP를 복원 시키기 위함) prologue에서 매개변수로 받아서 FP를 갱신하는데 생각해보니 프롤로그 함수 내에서 하면 func함수 내부를 최대한 안건드는 선에서 할 수 있을 것 같다. (FP가 전역변수이기 때문!)

라고 생각했으나.. 에필로그 전에 다음 func함수를 call하기 때문에 FP가 갱신되므로 그전에 FP를 저장해야하는 것을 깨달았다..

그럼 지금까지 내가 짠 함수들을과 func내부의 코드를 정리해보겠다.

1. push

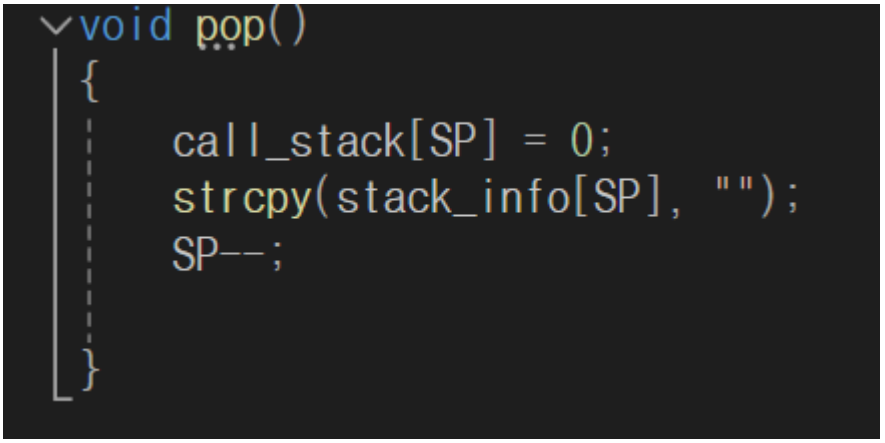


인자로 call_stack에 저장할 value(실제 변수에 저장되어 있는 값과 함수 이름, 매개, 지역변수의 이름을 stack_info에 저장할 value_name을 받는다.

그리고 SP가 Stack_size보다 작을 시 SP를 증가 시킨다. (Stack에 push 하므로)

그다음 차례대로 call_stack, stack_info에 저장을 한다 이때 value_name을 배열로 정의했으므로 strcpy를 사용하였다.

2. pop



pop은 인자로 무엇을 받지 않는다. 저장하는게 아니라 삭제 시키기 때문이다. call_stack과 stack_info 0으로 바꿔주고 SP를 낮춘다.

3. Prologue

```
void Prologue(char* func_name, int count, ...)
{
    int bfFP = FP;

    int* bf_FP = &bfFP;//이전 FP
    va_list args;
    va_start(args, count);
    for (int i = count-1; i >=0; i--) //함수 인자 stack에 push
    {
        int arg = va_arg(args, int);
        char temp[50];
        sprintf(temp, "arg%d", i+1);
        push(arg, temp);



    }
    va_end(args);

    push(-1, "return address");//return address push


    char sfp_info[50];
    sprintf(sfp_info, " %s SFP", func_name); //SFP push
    FP = SP;
    ++FP;
    push(*bf_FP, sfp_info);

}
```

(1). 매개 변수: stack_info에 저장될 함수 이름, 함수의 매개변수 인자 수를 저장할 count, ...⇒ 함수의 매개변수를 받기 위함(유동적임)

(2). 매개변수 push

처음에는 아직 FP가 이전 함수의 FP를 가르키고 있기 때문에 이를 저장할 새로운 변수를 만들고,

strdg.h 라이브러리에 있는 함수(va_list, va_start, va_arg를 이용하여 매개변수 값을 저장하고 sprintf을 이용해 arg1,arg2...등 매개변수의 이름을 배열로 만들어 저장한 후, push

(3). return address push

“return address” 문자열 자체가 stack_info에 저장된다.

(4). SFP push

함수 이름을 담을 sfp_info라는 새로운 배열을 선언후 매개변수와 마찬가지로 sprintf함수를 이용해 func_name을 저장한다 그리고 FP를 SP로 갱신 시킨 후 +1을 한다 그다음 이전 함수의 FP인 *bf_FP와 sfp_info 를 push 포인터로 한 이유는 뭔가 주소를 기억해서 돌아간다는 점이 포인터로 구현해야할 것같아서이다.

(5). 지역 변수 push

그다음 func함수 내애서 지역 변수 push를 해준다.

4.func함수

```
  //func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    Prologue("func1", 3,arg3, arg2, arg1);


    int FP1 = FP; //복원할 func1의 SFP 저장
    int* func1_SFP = &FP1;
    int var_1 = 100;
    push(var_1, "var_1");


    print_stack();



    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    Epilogue(1, 2, func1_SFP);

    print_stack();
}
```

에필로그에서 복원 할 FP를 저장한 후 prologue함수를 call한다. 이때 func1이므로 func_name=”func1” count=3(매개변수 3개) arg3, arg2, arg1 순서로 매개변수를 준

다. 매개변수 같은 경우 오른쪽부터 stack에 쌓이기 때문이다. 그 다음 지역변수 push , func2 call 후 에필로그 함수를 call 한다.

5. epilogue

```
void Epilogue(int count_local, int count_arg, int* SFP)
{
    for (int i = 0; i < count_local; i++)//지역 변수 pop
    {
        pop();
    }

    FP = *SFP; //SFP 복원
    pop();
    pop();//return address 제거
    for (int i = 0; i < count_arg; i++)
    {
        pop();
    }
}
```

(1). 매개변수: 지역 변수, 매개 변수의 수를 받는다⇒ 그 수만큼 pop을 해야하기 때문, 그리고 func 내부에 저장된 이전 함수 FP를 포인터로 받는다.

(2). 지역변수, 매개변수

반목문을 통해 pop

(3). FP갱신을 통해 SFP을 복원하고

SFP, return address를 차례대로 pop한다.