hw1

1. base1

```
/*  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)
    call_stack[++SP] = arg1;
    sprintf(stack_info[SP], "arg1");

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

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

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

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

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

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

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

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

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


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

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

    call_stack[++SP] = var_2;
    sprintf(stack_info[SP], "var_2");
    //////////////////////////////////////////////////
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    SP -= 1;
    SP--;
    SP--;
    SP = FP;
    FP = call_stack[SP];
    SP--;
    SP--;
    print_stack();
}


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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    call_stack[++SP] = arg1;
    sprintf(stack_info[SP], "arg1");
    call_stack[++SP] = -1;
    sprintf(stack_info[SP], "Return Address");
    call_stack[++SP] = FP;
    sprintf(stack_info[SP], "func3 SFP");
    FP = SP;
    call_stack[++SP] = var_3;
    sprintf(stack_info[SP], "var_3");
    call_stack[++SP] = var_4;
    sprintf(stack_info[SP], "var_4");

    //////
    print_stack();
```

```
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    SP -= 3;
    SP--;
    SP = FP;
    FP = call_stack[SP];
    SP--;
    SP--;

    print_stack();
    return 0;
}
```

func1

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

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    call_stack[++SP] = arg1;
    sprintf(stack_info[SP], "arg1");

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

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

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

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

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

- func1 스택 프레임 형성

1. 매개변수 arg1,arg2,arg3이 입력되었으므로 SP를 1씩 증가시키며 매개변수들을 스택에 저장한다
2. 매개변수 입력이 끝나고 나면 SP를 증가시키며 -1 을 입력해 return address 라는 것을 확인한다
3. SP를 1 증가시키고, FP를 이동하여 FP를 SP 위치에 갱신시켜 준다
4. 마지막으로 func1의 지역변수 var1을 스택 프레임에 입력한다

- func2 스택 프레임 제거

1. func2의 지역변수가 2개이므로 2만큼의 SP를 줄이고 SP를 FP 위치로 갱신한다
2. SFP의 데이터를 통해 FP를 갱신한다
3. func2의 매개변수의 경우 caller인 func1이 제거할 수 있도록 한다

func2

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

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

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

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


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

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

   call_stack[++SP] = var_2;
   sprintf(stack_info[SP], "var_2");
   //////////////////////////////////////////////////
   print_stack();
   func3(77);
   // func3의 스택 프레임 제거 (함수 에필로그 + pop)
   SP -= 1;
   SP--;
   SP--;
   SP = FP;
   FP = call_stack[SP];
   SP--;
   SP--;
   print_stack();
}
```

- func2의 스택 프레임 형성

1. sp를 증가시키며 func2의 매개변수 arg1,arg2 입력

2. 동일하게 -1을 넣어 return address를 표기한다
3. FP를 SP 위치에 갱신하며 SFP를 설정한다
4. 지역변수 var_2를 삽입한다

- func3의 스택 프레임 제거

1. func3의 지역변수가 2개이므로 -1을 2번 처리하고 다음 위치로 이동한다
2. SP를 FP값으로 변경한다
3. SFP를 통해 FP 갱신
4. func3의 매개변수를 제거할 수 있도록 한다

func3

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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    call_stack[++SP] = arg1;
    sprintf(stack_info[SP], "arg1");
    call_stack[++SP] = -1;
    sprintf(stack_info[SP], "Return Address");
    call_stack[++SP] = FP;
    sprintf(stack_info[SP], "func3 SFP");
    FP = SP;
    call_stack[++SP] = var_3;
    sprintf(stack_info[SP], "var_3");
    call_stack[++SP] = var_4;
    sprintf(stack_info[SP], "var_4");

    //////
    print_stack();
}
```

- func3의 스택 프레임 형성

1. func3의 매개변수인 arg1을 삽입
2. -1을 넣어 return address 표기. FP 설정
3. 지역변수 var_3,var_4 할당

main

```
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    SP -= 3;
    SP--;
    SP = FP;
    FP = call_stack[SP];
    SP--;
    SP--;
```

```
    print_stack();
    return 0;
}
```

- func1의 스택 프레임 제거

1. 지역변수 제거
2. SP갱신
3. FP 갱신

base1.c 실행 결과

```
racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/Cykor_week1/week_1$ ./base1
====== Current Call Stack ======
5 : var_1 = 100    <== [esp]
4 : func1 SFP    <== [ebp]
3 : Return Address
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================

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

====== 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 : arg2 = 13
6 : arg1 = 11
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================

====== Current Call Stack ======
11 : arg1 = 77    <== [esp]
```

```
10 : var_2 = 200
9 : func2 SFP = 4    ⇐= [ebp]
8 : Return Address
7 : arg2 = 13
6 : arg1 = 11
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================

====== Current Call Stack ======
7 : arg2 = 13    ⇐= [esp]
6 : arg1 = 11
5 : var_1 = 100
4 : func1 SFP    ⇐= [ebp]
3 : Return Address
2 : arg3 = 3
1 : arg2 = 2
0 : arg1 = 1
================================

====== Current Call Stack ======
2 : arg3 = 3    ⇐= [esp]
1 : arg2 = 2
0 : arg1 = 1
================================
```

base1.c 의 문제점

1. caller가 제대로 호출한 함수의 매개변수를 정리해 주지 못하면서 매개변수가 존치하게 된다
2. 매개변수의 오른쪽부터 저장이 되어야 하는데 순서가 반되가 되어있다
3. push,pop 등의 과정을 함수로 구현하지 않고 내부에서 작동하기 때문에 수정 및 시각적 확인이 어렵다

---

base2.c

```
/*  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);

// 함수 설정
void push_stack(int value, const char* info)
{
   call_stack[++SP] = value;
   sprintf(stack_info[SP], "%s", info);
}


void pop_stack(int count)
{
   SP -= count;
}



void restore_fp()
{
   SP = FP;
   FP = call_stack[SP];
   SP--;
}


/*
   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)
    push_stack(arg3, "arg3");
    push_stack(arg2, "arg2");
    push_stack(arg1, "arg1");
    push_stack(-1, "Return Address");
    push_stack(FP, "func1 SFP");
    FP = SP;
    push_stack(var_1, "var_1");
////////////////////////////////////////////////////////////////////
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(1);
    restore_fp();
    pop_stack(3);
    print_stack();
}


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

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

    push_stack(arg2, "arg2");
    push_stack(arg1, "arg1");
```

```
    push_stack(-1, "Return Address");
    push_stack(FP, "func2 SFP");
    FP = SP;
    push_stack(var_2, "var_2");
    //////////////////////////////////////////////////
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(2);
    restore_fp();
    pop_stack(2);

    print_stack();
}


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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    push_stack(arg1, "arg1");
    push_stack(-1, "Return Address");
    push_stack(FP, "func3 SFP");
    FP = SP;
    push_stack(var_3, "var_3");
    push_stack(var_4, "var_4");
    //////
    print_stack();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(1);
    restore_fp();
    pop_stack(4);
    print_stack();
    return 0;
}
```

push,pop, restore

```
void push_stack(int value, const char* info)
{
    call_stack[++SP] = value;
    sprintf(stack_info[SP], "%s", info);
}


void pop_stack(int count)
{
    SP -= count;
```

```
}



void restore_fp()
{
    SP = FP;
    FP = call_stack[SP];
    SP--;
}
```

→ 효율적인 함수 프롤로그, 에필로그 구성을 위해서 push,pop 함수를 구현했다

또한 FP를 복원하는 과정 또한 함수로 저장했다

SP를 FP 위치로 갱신, SFP를 통해 FP 갱신, SP 감소 과정을 함수로 정리하였다

func1

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

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    push_stack(arg3, "arg3");
    push_stack(arg2, "arg2");
    push_stack(arg1, "arg1");
    push_stack(-1, "Return Address");
    push_stack(FP, "func1 SFP");
    FP = SP;
    push_stack(var_1, "var_1");
////////////////////////////////////////////////////////////////////
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(1);
    restore_fp();
    pop_stack(3);
    print_stack();
}
```

→ 위에서 작성한 push_stack,pop_stack을 통하여 base1.c 의 각 함수를 변경한다

1. 매개변수 push
2. -1로 표기하며 return address 표기
3. SFP 표시 → sp를 증가시키며 FP를 해당 위치에 저장
4. 지역변수 삽입

- func2의 스택 프레임 제거 또한 base1.c와 동일하게 수행한다

→ func2의 지역변수 1개이므로 pop_stack(1)

→restore_fp() 이후, return addres에 위치한 sp를 감소시켜 매개변수 제거

func2

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

```
    int var_2 = 200;

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

    push_stack(arg2, "arg2");
    push_stack(arg1, "arg1");
    push_stack(-1, "Return Address");
    push_stack(FP, "func2 SFP");
    FP = SP;
    push_stack(var_2, "var_2");
    ////////////////////////////////////////////////
    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(2);
    restore_fp();
    pop_stack(2);

    print_stack();
}
```

- 함수 프롤로그의 경우 func1와 동일하다
- func3의 스택 프레임 제거

1. func3의 지역변수가 2개이므로 pop_stack(2), restore_fp()
2. 매개변수가 1개이므로 고려해서 pop

func3

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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    push_stack(arg1, "arg1");
    push_stack(-1, "Return Address");
    push_stack(FP, "func3 SFP");
    FP = SP;
    push_stack(var_3, "var_3");
    push_stack(var_4, "var_4");
    //////
    print_stack();
}
```

- func3의 스택 프레임의 경우 func1와 동일한 과정을 수행한다

main

```
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    pop_stack(1);
```

```
    restore_fp();
    pop_stack(4);
    print_stack();
    return 0;
}
```

- func1의 스택 프레임 제거

1. func1의 지역변수는 1개 이므로 제거
2. Fp 복원
3. func1의 매개변수는 3개이므로 고려해서 pop 처리

base2.c 결과

```
racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/Cykor_week1/week_1$ ./base2
====== 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    ⇐= [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.
```

올바르게 매개변수의 오른쪽부터 나타나는 것을 볼 수 있고, 호출한 함수의 매개변수 arg가 정상적으로 지워진 것을 확인할 수 있다

base2.c 개선점

1. 함수 에필로그, 프롤로그 과정이 명확하게 보이지 않고 push,pop 등이 반복해서 사용된다

→ 함수 에필로그 , 프롤로그 과정을 함수화 하여 보다 명확하게 표기한다

---

base3.c

```
#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 문자열 배열

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

// 스택 조작 함수
void push_stack(int value, const char* info)
{
   call_stack[++SP] = value;
   sprintf(stack_info[SP], "%s", info);
}
```

```
void pop_stack(int count)
{
    SP -= count;
}

void restore_fp()
{

    FP = call_stack[SP];
    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");
}

// 공통 프레임 생성 함수
void create_frame(int argc, int arg[], const char* argnames[],
                  const char* sfp,
                  int localc, int local[], const char* localnames[])
{
    for (int i = argc - 1; i >= 0; i--)
        push_stack(arg[i], argnames[i]);
    push_stack(-1, "Return Address");
    push_stack(FP, sfp);
    FP = SP;
    // 4. 지역변수 공간 확보
    SP += localc;

    // 5. FP 기준으로 지역변수 값 채우기
    for (int i = 0; i < localc; i++) {
        int index = FP + 1 + i;
        call_stack[index] = local[i];
        sprintf(stack_info[index], "%s", localnames[i]);
    }
}
```

```
// 공통 프레임 제거 함수
void remove_frame(int localc, int argc)
{
    SP = FP;     // 지역변수 제거
    restore_fp();          // FP 복원
    pop_stack(argc + 1);    // 매개변수 + Return Address 제거
}

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

// func1 정의

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

    int argv[] = {arg1, arg2, arg3};
    const char* argnames[] = {"arg1", "arg2", "arg3"};
    int localv[] = {var_1};
    const char* localnames[] = {"var_1"};

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    create_frame(3, argv, argnames, "func1 SFP", 1, localv, localnames); //func1 : 매개변수 3개, 지역변수 1개개
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    remove_frame(1, 2); // 지역변수 1개, 매개변수 2개
    print_stack();
}

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

    int argv[] = {arg1, arg2};
    const char* argnames[] = {"arg1", "arg2"};
    int localv[] = {var_2};
    const char* localnames[] = {"var_2"};

    create_frame(2, argv, argnames, "func2 SFP", 1, localv, localnames); //func2 생성 : 매개변수 2개, 지역변수 1개
    print_stack();

    func3(77);

    remove_frame(2, 1); //func3제거 : 지역 2개, 매개변수 1개
    print_stack();
}

// func3 정의
void func3(int arg1)
{

    int var_3 = 300;
    int var_4 = 400;
```

```
    int argv[] = {arg1};
    const char* argnames[] = {"arg1"};
    int localv[] = {var_3, var_4};
    const char* localnames[] = {"var_3", "var_4"};

    create_frame(1, argv, argnames, "func3 SFP", 2, localv, localnames); //매개변수 1개, 지역변수 2개
    print_stack();
}

// main 함수
int main()
{
    func1(1, 2, 3);

    // func1 프레임 제거
    remove_frame(1, 3);  // 매개변수 3개, 지역변수 1개
    print_stack();
    return 0;
}
```

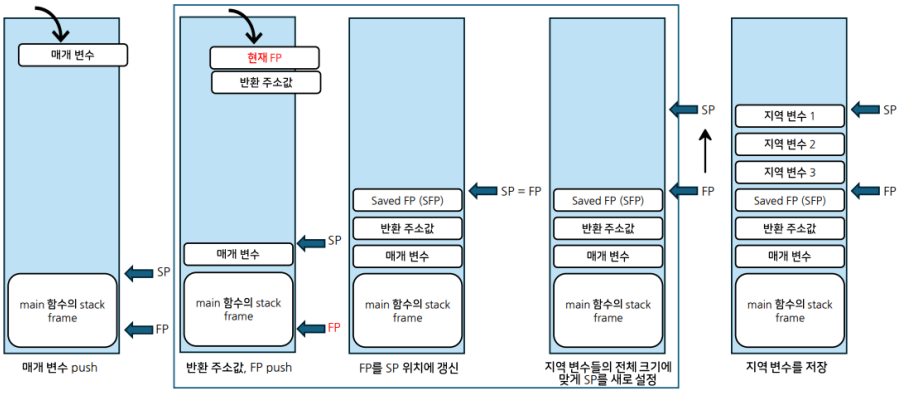
스택 조작 관련 함수

```
void push_stack(int value, const char* info)
{
    call_stack[++SP] = value;
    sprintf(stack_info[SP], "%s", info);
}

void pop_stack(int count)
{
    SP -= count;
}

void restore_fp()
{

    FP = call_stack[SP];
    SP--;
}
```

creat_stack frame

```
void create_frame(int argc, int arg[], const char* argnames[],
                  const char* sfp,
                  int localc, int local[], const char* localnames[])
{
    for (int i = argc - 1; i >= 0; i--)
        push_stack(arg[i], argnames[i]);
    push_stack(-1, "Return Address");
    push_stack(FP, sfp);
    FP = SP;
    // 4. 지역변수 공간 확보
    SP += localc;
```

```
    // 5. FP 기준으로 지역변수 값 채우기
    for (int i = 0; i < localc; i++) {
        int index = FP + 1 + i;
        call_stack[index] = local[i];
        sprintf(stack_info[index], "%s", localnames[i]);
    }
}
```

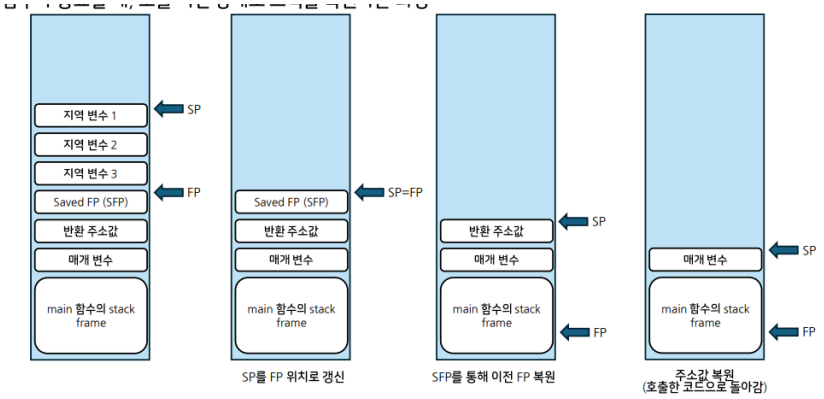


1. 매개변수 : 함수의 매개변수 개수, 함수의 매개변수, SFP,지역변수 등을 설정
2. argc : 매개변수 개수만큼 stack을 push하며 매개변수 데이터 입력
3. 반환 주소값 ,SFP push
4. FP를 SP 위치에 갱신
5. 지역변수 공간 확보
6. FP를 기준으로 지역변수들을 저장

→ 함수 프롤로그의 작동 과정 순서에 맞추어서 실재로 스택에 저장되는 것을 나타낼 수 있다

remove_frame

```
void remove_frame(int localc, int argc)
{
    SP = FP;     // 지역변수 제거
    restore_fp();          // FP 복원
    pop_stack(argc + 1);    // 매개변수 + Return Address 제거
}
```



1. SP를 FP 위치로 갱신하며 지역변수 제거
2. SFP를 통해 FP 복원
3. c프로그램이므로 caller에서 호출한 매개변수를 제거할 수 있도록 매개변수 값만큼 스택에서 pop 처리

→ 이를 통해 함수 에필로그 작동 과정을 표현 가능하다

func1

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

   int argv[] = {arg1, arg2, arg3};
   const char* argnames[] = {"arg1", "arg2", "arg3"};
   int localv[] = {var_1};
   const char* localnames[] = {"var_1"};

   // func1의 스택 프레임 형성 (함수 프롤로그 + push)
   create_frame(3, argv, argnames, "func1 SFP", 1, localv, localnames); //func1 : 매개변수 3개, 지역변수 1개개
   print_stack();
   func2(11, 13);
   // func2의 스택 프레임 제거 (함수 에필로그 + pop)
   remove_frame(1, 2); // 지역변수 1개, 매개변수 2개
   print_stack();
}
```

- func1의 매개변수가 3개, 지역변수가 1개 이므로 해당 값들을 create_frame에 설정
- func2의 스택 프레임 제거 : 지역변수 1개 ,매개변수 2개 이므로 해당 값들을 함수에 입력

func2

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

   int argv[] = {arg1, arg2};
   const char* argnames[] = {"arg1", "arg2"};
   int localv[] = {var_2};
   const char* localnames[] = {"var_2"};

   create_frame(2, argv, argnames, "func2 SFP", 1, localv, localnames); //func2 생성 : 매개변수 2개, 지역변수 1개
   print_stack();

   func3(77);

   remove_frame(2, 1); //func3제거 : 지역 2개, 매개변수 1개
   print_stack();
}
```

- func2는 매개변수 2개, 지역변수 1개 이므로 해당 값들을 create_frame 한다
- func3은 지역변수 2개, 매개변수 1개 이므로 해당 값들로 remove_frame 처리한다

func3

```
void func3(int arg1)
{

    int var_3 = 300;
    int var_4 = 400;


    int argv[] = {arg1};
    const char* argnames[] = {"arg1"};
    int localv[] = {var_3, var_4};
    const char* localnames[] = {"var_3", "var_4"};

    create_frame(1, argv, argnames, "func3 SFP", 2, localv, localnames); //매개변수 1개, 지역변수 2개
    print_stack();
}
```

- func3 스택 프레임 생성 : 매개변수 1개, 지역변수 2개이므로 해당 값들로 스택 프레임 생성

main

```
int main()
{
    func1(1, 2, 3);

    // func1 프레임 제거
    remove_frame(1, 3);  // 매개변수 3개, 지역변수 1개
    print_stack();
    return 0;
}
```

- func1 프레임 제거 : 매개변수 3개, 지역변수 1개 이므로 해당 값들로 func1 스택 프레임을 제거한다

base3.c 결과 분석

```
====== 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    ⇐= [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.
```

1. func1 스택 프레임 생성

```
====== Current Call Stack ======
5 : var_1 = 100    ⇐= [esp]
```

```
4 : func1 SFP    ⇐= [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================
```

void func1(int arg1, int arg2, int arg3) 이므로 arg3부터 값이 정상적으로

2. func2 스택 프레임 생성

```
====== 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
================================
```

3. func3 스택 프레임 생성

```
====== 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
================================
```

4. func3 스택 프레임 제거

```
====== 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
================================
```

5. func2 스택 프레임 제거

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

6. func1 스택 프레임 제거

```
================================

Stack is empty.
```