2025_1주차_과제_수정내용

스마트보안학부 2024350034 한주영

```
수정해야 하는 내용
수정한 코드 내용
출력 결과
```

수정해야 하는 내용

1. 내가 혼자 보기 편하라고 만든 Saved FP를 피피티에 나와있는대로 funcX SFP로 수정 → **funcX SFP로는 수정 못하고 SFP로 수정완**
2. arg_X로 코딩 작성을 해두었는데 피피티에 안내되어있는대로 그냥 argX로 수정 → **수정완**
3. 매개변수, 지역변수 코드 작성 모두 수정 필요 → **수정완**
4. 피피티 결과값에 알맞게 수정 → **수정완**

수정한 코드 내용

매개변수는 쉽게 for문 조건 수정으로 알맞게 결과값을 출력할 수 있었다. 하지만 지역변수의 경우 num_locals가 아래 func1,2,3에 유기적으로 연결되어 있지 못해 결과값이 계속 var_1=400이 나온다거나 오류가 출력되었다. 따라서 이를 유기적으로 잘 연결하기 위하여 전역변수에 지금까지 푸시한 지역변수 총 개수를 카운트하도록 설정했다. 이렇게 함으로써 기존에 작성한 코드에서 num_locals 조건을 설정하는 데 편했다.

문제는 하나를 수정하면 줄이 갑자기 더 늘어나고 또 다른 하나를 수정하면 var_4가 갑자기 300이 되면서 코드를 계속 수정했다. 그렇다보니 Saved FP를 피피티에 나와있는 funcX SFP를 그대로 완벽하게 구현해내지는 못했지만 결과적으로 SFP가 정상적으로 출력되게는 구현을 성공적으로 하였다.

```
/*  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"
    ============================================================================
*/

// 스마트보안학부 2024350034 한주영

#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;
static int local_offset = 0;

// push, pop 함수
void push(int value, const char* desc) {
    SP++;
    call_stack[SP] = value;
    strcpy(stack_info[SP], desc);
}
void pop(int pop_num) {
    SP -= pop_num;
}

// 함수 프롤로그
void prologue(int args[], int num_args, int locals[], int num_locals) {
    // 매개변수 push_args
    for (int i = num_args - 1; i >= 0; --i) {
        char label[6];
        strcpy(label, "arg");
        label[3] = '1' + i;
        label[4] = '\0';
        push(args[i], label);
    }
    // 반환 주소 저장
    push(-1, "Return Address");
    // 이전 FP 저장 & 새 FP 설정
    push(FP, "SFP");
    FP = SP;
    // 지역변수 push_local
    for (int i = 0; i < num_locals; i++) {
        char label[6];
        strcpy(label, "var_");
        int idx = local_offset + i + 1;
        label[4] = '0' + idx;
        label[5] = '\0';
        push(locals[i], label);
    }
    local_offset += num_locals;
}
// 함수 에필로그
void epilogue(int pop_num) {
    pop(pop_num);
    FP = call_stack[SP + 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)
   int args[] = {arg1, arg2, arg3};
   int locals[] = {var_1};
   prologue(args, 3, locals, 1);
   print_stack();

   func2(11,13);

   // func2의 스택 프레임 제거 (함수 에필로그 + pop)
   epilogue(5);
   print_stack();
}


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

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

```
    int args[] = {arg1, arg2};
    int locals[] = {var_2};
    prologue(args, 2, locals, 1);
    print_stack();

    func3(77);

    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    epilogue(5);
    print_stack();
}


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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    int args[] = {arg1};
    int locals[] = {var_3, var_4};
    prologue(args, 1, locals, 2);
    print_stack();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    epilogue(6);
    print_stack();
    return 0;
}
```

출력 결과

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

====== Current Call Stack ======
10 : var_2 = 200    <=== esp
9 : SFP = 4    <=== ebp
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : 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 : SFP = 9    <=== ebp
12 : Return Address
11 : arg1 = 77
10 : var_2 = 200
9 : SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================
```

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

====== Current Call Stack ======
5 : var_1 = 100    <=== esp
4 : SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

Stack is empty.
```