Callstack_2022350217_류대호

```
int garbage = 0; // pop한 값을 사용하지않을 때 garbage에다가 짬처리
void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);
void push(int a);
void pop(int *a);


void push(int a){
   SP++;
   call_stack[SP] = a;
}

void pop(int *a){
   *a = call_stack[SP];
   SP--;
}
```

어셈블리에서와 같이 push, pop함수를 구현했습니다.

과제에서 스택의 최하단을 -1로 설정해놨으므로

스택이 쌓일수록 값을1씩 늘려가므로

push함수는 sp를 하나 올려주고,

인자로 전달받은 a를 stack값으로 저장해줍니다.

pop은 그의 역과정으로 진행됩니다.

stack최상단에서 값을 뽑아서, 원하는 변수에 값을 저장해줍니다.(FP의 값을 복원할때 쓸 예정)

포인터 변수 a에 주소값을 전달해주면,

스택 최상단, call_stack[SP]의 값을 저장해줍니다.

또한 pop한 값을 사용하지 않고싶다면, garbage에다가 저장을 해줍니다

```
//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(arg3); strcpy(stack_info[SP],"arg3");
   push(arg2); strcpy(stack_info[SP],"arg2");
   push(arg1); strcpy(stack_info[SP],"arg1");

   push(-1); strcpy(stack_info[SP],"Return Address");
   push(FP); strcpy(stack_info[SP],"func1's SFP");
   FP = SP;
   push(var_1); strcpy(stack_info[SP],"func1's var_1");



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

caller가 매개변수를 오른쪽부터 스택에 push하고

prolog단계에 진입합니다

return address와 fp를 push합니다.

push를 하면서

strcpy를 통해서 stack_info에 변수명을 저장합니다

매개변수와 RETURN_ADDRESS, 를 push해주고나서,

func1의 FP를 push해줍니다

이제 FP를 업데이트 해주고,

func2를 호출합니다.

```
/Users/firenabang2012/Documents/Cykor_2025/Cykor_system1_week1/cmake-build-debug/Cykor_week1
====== Current Call Stack ======
5 : func1's var_1 = 100    <=== [esp]
4 : func1's SFP    <=== [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================
```

func1내에서 printstack을 한 결과입니다.

값들이 제대로 반영되었군요

같은방식으로 func2, func3에서의 프롤로그를 구현해주겠습니다.

```
/*  call_stack

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

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

    ============================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
#define RETURN_ADDRESS -1 // return address를 -1로 놓기로 가정

int     call_stack[STACK_SIZE];         // Call Stack을 저장하는 배열
char    stack_info[STACK_SIZE][20];     // Call Stack 요소에 대한 설명을 저장하는

/*  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[

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;
int ret = -1;
int garbage = 0; // pop한 값을 사용하지않을 때 garbage에다가 짬처리
void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);
void push(int a);
void pop(int *a);
```

```
void push(int a){
    SP++;
    call_stack[SP] = a;
}

void pop(int *a){
    *a = 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(arg3); strcpy(stack_info[SP],"arg3");
    push(arg2); strcpy(stack_info[SP],"arg2");
    push(arg1); strcpy(stack_info[SP],"arg1");

    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func1's SFP");
    FP = SP;

    SP++; //지역변수 저장할 공간 만들기
    call_stack[SP] = var_1; strcpy(stack_info[SP],"var_1");



    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)
    push(arg2); strcpy(stack_info[SP],"arg2");
    push(arg1); strcpy(stack_info[SP],"arg1");

    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func2's SFP");
    FP = SP;
    SP++; //지역변수 공간 할당
    call_stack[SP] = var_2;  strcpy(stack_info[SP],"var_2");

    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)
    push(arg1); strcpy(stack_info[SP],"arg1");
    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func3's SFP");
    FP = SP;
    SP+=2; //지역변수 공간 할당
    call_stack[SP] = var_4;
    call_stack[SP -1] = var_3;
    push(var_3); strcpy(stack_info[SP],"var_3");
    push(var_4); strcpy(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)
    print_stack();
    return 0;
}
```

```
/Users/firenabang2012/Documents/Cykor_2025/Cykor_system1_week1/cmake-build-debug/Cykor_week1
====== Current Call Stack ======
5 : var_1 = 100    <=== [esp]
4 : func1's SFP    <=== [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : func2's SFP = 4    <=== [ebp]
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1's 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's SFP = 9    <=== [ebp]
12 : Return Address
```

잘 동작합니다.

2. 함수 에필로그

이제 함수 에필로그를 구현해 주겠습니다.

앞서 구현했던 pop함수를 토대로 구현을 해보았습니다.

```
/*  call_stack

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

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

   ============================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
#define RETURN_ADDRESS -1 // return address를 -1로 놓기로 가정

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]` 

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;
int ret = -1;
int garbage = 0; //pop한 값을 사용하지 않을 때 garbage에다가 저장

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);
void push(int a);
void pop(int *a);


void push(int a){
   SP++;
   call_stack[SP] = a;
}

void pop(int *a){
   *a = 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(arg3); strcpy(stack_info[SP],"arg3");
    push(arg2); strcpy(stack_info[SP],"arg2");
    push(arg1); strcpy(stack_info[SP],"arg1");
```

```
    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func1's SFP");
    FP = SP;
    SP++;//지역변수 공간 할당
    call_stack[SP] = var_1; strcpy(stack_info[SP],"var_1");



    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    pop(&FP);
    pop(&garbage); //return address 및 매개변수 제거
    pop(&garbage);
    pop(&garbage);

    print_stack();

}


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

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg2); strcpy(stack_info[SP],"arg2");
    push(arg1); strcpy(stack_info[SP],"arg1");

    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func2's SFP");
    FP = SP;
    SP++;//지역변수 공간 할당
    call_stack[SP] = var_2; strcpy(stack_info[SP],"var_2");
    print_stack();
```

```
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    pop(&FP);
    pop(&garbage); //return address 및 매개변수 제거
    pop(&garbage);

    print_stack();

}

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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg1); strcpy(stack_info[SP],"arg1");
    push(-1); strcpy(stack_info[SP],"Return Address");
    push(FP); strcpy(stack_info[SP],"func3's SFP");
    FP = SP;
    SP+=2; //지역변수 공간 할당
    call_stack[SP] = var_4; strcpy(stack_info[SP],"var_4");
    call_stack[SP -1] = var_3; strcpy(stack_info[SP -1],"var_3");

    print_stack();
}


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

    SP = FP;
```

```
    pop(&FP);
    pop(&garbage); //return address 및 매개변수 제거
    pop(&garbage);
    pop(&garbage);
    pop(&garbage);

    print_stack();
    return 0;
}
```

SP를 FP위치로 옮기고난다음

스택 최상단의 return address값을 통해서 ret했다고 치고(pop eip느낌으로 pop (&garbage))

pop(&garbage)를 통해서 매개변수를 정리해줍니다

caller가 callee에 전달한 매개변수를 정리한 시점에

실행결과는

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

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

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

Stack is empty.

Process finished with exit code 0
```