<과제 결과>

```
====== 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.
```

<1th commit>

주요 내용 : push() 함수의 개략적 구현

1) push() 함수를 초기적으로 구현하여, 스택에 데이터를 쌓는 구조의 기초를 설정하였다..

<2th commit>

주요 내용 : 문자열 처리 및 함수 개선

1) 포인터 변수 c : char c[10]에서 char *c로 변경하여 포인터를 사용하였다. 배열로 설정하여 한 문자씩 받는 방식을 수정하고자 하였고, 이를 통해 한 번에 문자를 받을 수 있게 되었다.
2) strncpy() 함수 : for문으로 개별 문자를 복사하는 방식을 수정하고자 하였다. 이를 strncpy() 함수로 대체하여 코드의 가독성과 안정성을 향상시켰다.
3) 매개변수 a를 value로 변수명을 변경하여 의미를 명확하게 하였다.
4) push(“arg x”, arg x)의 형식으로 호출 구조를 일관되게 통일하였다.

<3th commit>

주요 내용 : 실행 오류 수정

1) 문자열을 복사할 때 널문자가 누락되어 오류가 발생하던 부분을 해결하기 위해 #define _CRT_SECURE_NO_WARNINGS를 추가하였고, 배열 끝에 '₩0'을 명시적으로 추가함으로써 문자열의 종료를 보장하였다.

<4th commit>

주요 내용 : FP 구현

1) FP의 구현 : 스택에 Return Address가 쌓이는 시점을 기준으로 잡았다. 그 다음 push될 스택은 SFP이므로 Return Address가 쌓인 시점에서 FP = SP +1로 두면 SFP가 쌓일 위치에서 FP = SP가 되기 때문이다. 이는 함수 호출 직후 esp를 ebp에 복사하는 실제 호출 구조를 반영한 것이다. printf("%d₩n", FP);는 FP가 제대로 갱신되는지 확인하기 위해 잠깐 추가했던 것이다.

<5th commit>

주요 내용 : temp 배열 추가 및 pop() 함수 초안 작성

1) temp 배열 : 이전까지는 숫자로 명시하였던 SFP의 위치를 동적으로 표현하기 위해 temp 배열을 추가하였다. 각 함수의 호출 시점에 현재 FP는 temp배열에 저장되고, 새로운 FP는 현재 스택 위치(SP)로 갱신된다. 이 과정들을 통해 각 스택 프레임들은 SFP를 통해 이전 프레임과 연결되며 프레임 간 링크드 리스트 구조를 형성한다.
2) pop() 함수 : 개략적으로 구현하였다. temp 사이의 간격으로 프레임 크기를 유추하여 반복문으로 SP를 줄이는 방식을 이용하였다.

<6th commit>

주요 내용 : 크기 추적 방식 변경 및 pop() 함수의 개선

1) frame_size 배열 추가 : pop() 함수에서 temp[i] – temp[i-1]로 프레임 크기를 유추하는 방법을 수정하고자 하였다. 각 함수의 스택이 끝나면 frame_size에 각 함수의 크기를 저장한다. 이전보다 더 정확하게 계산할 수 있다.
2) pop() 함수 수정 : 반복문을 없애고 SP -= frame_size[i]로 보다 간단하게 만들 수 있긴 하다. 반복문이 비효율적이라고 생각하여 수정하려고 했으나, FP가 갱신되는 시점을 보다 명확하게 구현하고 싶었기 때문에 반복문의 구조를 유지하였다. SP가 FP -1, 즉 Return Address가 되는 시점에 FP를 이전 스택의 SFP 위치로 옮긴다.

<7th commit>

주요 내용 : 코드 정리 및 안정성 강화

1) pop() 함수 내에서만 사용되는 변수 j를 지역 변수로 넣어 범위 제한을 명확하게 하였다.
2) frame_start_sp[] 배열 : 프레임 시작 위치를 추적할 수 있도록 개선하여 이전에는 복잡하게 이어졌던 frame_size의 계산을 일관되게 수행할 수 있게 수정하였다. 수정 과정에서 중복 로직을 정리하여 calculate_frame_start() 함수 및 calculate_frame_size() 함수로 묶어 가독성을 높이고자 하였다.
3) 배열 크기를 STACK_SIZE로 통일하여 코드 유지보수성을 향상시켰다.
4) "Return Address" 문자열을 #define RETURN_ADDR로 정의하여 오타 가능성을 방지하고 코드 안정성을 높였다.
5) 문자를 20자까지 복사하고 21번째(배열상 20번째 인덱스)에 널문자를 추가한 것이 문제가 될 수도 있기 때문에 범위 내 삽입(stack_info[SP][19])으로 수정하여 버퍼 오버런을 방지하였다.