push 함수 작성

먼저 push 기능을 구현하기 위해 아래와 같이 함수를 작성하였다.

```
void push(int value, char *info)
{
    int length = strlen(info); // 설명문의 길이
    if (SP == STACK_SIZE - 1) // 스택이 가득 찬 경우
    {
        printf("Stack is full.\n");
        return;
    }
    if (length > 20) // 설명문이 20자를 초과하는 경우
    {
        printf("The length of info is up to 20 characters.\n");
        return;
    }

    SP++;
    call_stack[SP] = value;
    for (int i = 0; i<length; i++)
    {
        stack_info[SP][i] = info[i];
    }
}
```

먼저 입력받은 설명문의 길이를 저장해두고, 스택의 최대 크기가 50으로 주어졌기에 이를 넘으려할 경우 오류 메세지를 출력하는 코드를 작성했다.

설명문의 길이 또한 20자로 제한되어 있었기에 설명문의 길이를 검사하는 코드를 추가하고, 위 검사 코드를 모두 통과한 경우 SP를 증가시키고 입력받은 내용을 스택에 쌓는다.

func1, func2, func3 프롤로그 구현

위에서 작성한 push 함수를 바탕으로 func1, func2, func3 함수들의 프롤로그를 아래와 같이 작성했다.

```
// func1의 스택 프레임 형성 (함수 프롤로그 + push)
push(arg3, "arg3"); // arg3 push
push(arg2, "arg2"); // arg2 push
push(arg1, "arg1"); // arg1 push
push(-1, "Return Address"); // Return Address push
push(FP, "func1 SFP"); // Saved Frame Pointer push
FP = SP; // 현재 스택 프레임 포인터를 SP로 설정
push(var_1, "var_1"); // var_1 push
```

```
// func2의 스택 프레임 형성 (함수 프롤로그 + push)
push(arg2, "arg2"); // arg2 push
push(arg1, "arg1"); // arg1 push
push(-1, "Return Address"); // Return Address push
push(FP, "func2 SFP"); // Saved Frame Pointer push
FP = SP; // 현재 스택 프레임 포인터를 SP로 설정
push(var_2, "var_2"); // var_2 push
```

```
// func3의 스택 프레임 형성 (함수 프롤로그 + push)
push(arg1, "arg1"); // arg1 push
push(-1, "Return Address"); // Return Address push
push(FP, "func3 SFP"); // Saved Frame Pointer push
FP = SP; // 현재 스택 프레임 포인터를 SP로 설정
push(var_3, "var_3"); // var_3 push
push(var_4, "var_4"); // var_4 push
```

먼저 매개변수를 순차적으로 push하고 반환 주소값과 FP를 push한뒤 FP를 SP위치에 갱신했다.

이후 원칙적으로는 지역변수들의 크기에 맞게 SP가 한번에 조정되고 지역변수가 대입되어야 하지만, 위 내용을 코드로 작성하기에는 직관성이 떨어져 지역변수 저장 또한 push로 구현하였다.

pop 함수 작성

스택을 쌓는 과정은 모두 구현이 되었으니 함수의 에필로그를 구현하기에 앞서 pop 함수를 아래와 같이 작성하였다.

```
void pop()
{
    if (SP == -1) // 스택이 비어있는 경우
    {
        printf("Stack is empty.\n");
        return;
    }

    SP--;
}
```

스택이 이미 비어있는 상태에서 pop을 시도할 경우 오류 메세지를 출력하는 코드를 먼저 작성하고, 위 조건에 해당이 되지 않을 경우 SP를 낮추는 방식으로 pop을 구현하였다.

func1, func2, func3 에필로그 구현

위에서 작성한 pop 함수를 바탕으로 func1, func2, func3 함수들의 에필로그를 아래와 같이 작성했다.

```
// func3의 스택 프레임 제거 (함수 에필로그 + pop)
SP = FP; // SP를 FP 위치로 갱신
FP = call_stack[SP]; // SFP를 통해 이전 FP로 복원
pop(); // pop func3 SFP
pop(); // pop Return Address
pop(); // pop arg1
```

```
// func2의 스택 프레임 제거 (함수 에필로그 + pop)
SP = FP; // SP를 FP 위치로 갱신
FP = call_stack[SP]; // SFP를 통해 이전 FP로 복원
pop(); // pop func2 SFP
pop(); // pop Return Address
pop(); // pop arg1
pop(); // pop arg2
```

```
// func1의 스택 프레임 제거 (함수 에필로그 + pop)
SP = FP; // SP를 FP 위치로 갱신
FP = call_stack[SP]; // SFP를 통해 이전 FP로 복원
pop(); // pop func1 SFP
pop(); // pop Return Address
pop(); // pop arg1
pop(); // pop arg2
pop(); // pop arg3
```

함수 에필로그의 방식을 그대로 따라 SP를 FP 위치로 갱신하고 그 위치에 저장된 SFP를 통해 이전 FP로 복원했다.

이후 기존에 push를 통해 스택에 쌓여있던 데이터들을 순차적으로 pop해주며 스택 프레임을 제거했다.