week1

- func1, fun2, fun3에서 함수가 시작할때, 끝날 때 각각 함수의 프롤로그, 에필로그, pop, push를 구현해야한다.
- 이때 함수 내부에서 함수를 불러오지 않아 에필로그를 작성할 필요 없는 func3 먼저 코드를 작성해주었다.

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

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

    call_stack[++SP] = var_3;
    strcpy(stack_info[SP], "var_3");
    call_stack[++SP] = var_4;
    strcpy(stack_info[SP], "var_4");

    print_stack();
}
```

- 프롤로그 과정을 살펴보면 차례대로 스택에
  1. 매개변수 push
  2. 반환주소값, FP (SFP)을 push
  3. 지역 변수 크기에 맞게 SP 새로 설정
  4. 지역변수 저장(push)

  다음과 같은 과정인 것을 볼 수 있었다.

그러므로 call_stack[++FP] = arg1;이라는 방식을 통해 push를 구현해주었다. 먼저 매개변수인 arg1을 push해준뒤, 반환 주소, 현재FP를 저장(SFP)하기 위해 push해줬다. 그 뒤, FP = SP를 통해 SP값을 맞춰주었고, 지역변수들은 var_3, var4를 각각 push해주었다.

- 이때 위와 같이 코드를 작성하는 것보단, push함수와 pop함수를 따로 지정해 구현하는 것이 훨씬 가독성 좋다고 생각해 아래와 같이 push함수를 구현해준 뒤 func3함수에서 프롤로그 과정을 다시 만들어주었다.

```
void push(int element, char info[]){
    call_stack[++SP] = element;
    strcpy(stack_info[SP], info);
}
```

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

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)

    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func3 SFP");
    FP = SP;


    push(var_3, "var3");
    push(var_4, "var4");
    print_stack();
}
```

- 또한 pop함수 역시 만들어주었다.

```
void pop(){
    SP -= 1;
}
```

- func_3는 fun2함수에서 호출하고, func2함수는 func1함수에서 각각 호출하므로 각 함수에서 에필로그 과정을 만들어야 한다. 먼저 func_2함수를 살펴보자.

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

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

    push(arg2, "arg2");
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func2 SFP");
    FP = SP;

    push(var_2, "var3");

    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    FP = call_stack[SP--];
    pop();

    pop();
    print_stack();
}
```

- 프롤로그 부분은 func3와 동일하게 작성해주었고, 에필로그와 pop부분이 새롭게 들어온다
- 에필로그 과정을 살펴보면

  1. SP를 FP 위치로 갱신
  2. SFP를 통해 이전 FP위치 복원
  3. 주소값 복원

  이라는 단계를 통해 전개된다. 그러므로 먼저 SP = FP를 통해 FP의 위치를 갱신해준디. 이때 SP가 SFP의 위치를 가르키고 있기 때문에 FP = call_stack[SP—];을 통해 FP위치를 복원하면서 SP가 반환 주소값을 가르키

도록 만들었고, 그 뒤, pop을 한번 더 진행해주면서 SP가 매개변수 위치를 가르키도록 만들었다.

그 뒤 pop()함수를 호출했었던 함수(func3)의 매개변수 개수만큼 호출해주며 결과적으로 SP가 func2를 호출했었을 상태로 만들어주었다.

- func1함수에서의 프롤로그와 에필로그 과정을 위와 같은 방식으로 구현해줄 수 있었다.

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

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg3, "arg3");
    push(arg2, "arg2");
    push(arg1, "arg1");


    push(-1, "Return Address");
    push(FP, "func1 SFP");
    FP = SP;

    push(var_1, "var1");

    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    FP = call_stack[SP--];
    pop();

    pop();
    pop();
    print_stack();
}
```

- 그 뒤 main함수에서의 func1함수의 에필로그 과정을 다음과 같이 작성했다.

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

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

- 코드를 실행시켜본 결과 다음과 같은 결과들을 얻어낼 수 있었다.

```
[Running] cd "c:\Cykor\week1\Cykor_week1\" && gcc base.c -o base && "c:\Cykor\week1\Cykor_week1\"base
====== Current Call Stack ======
5 : var1 = 100    <=== [esp]
4 : func1 SFP    <=== [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==============================

====== Current Call Stack ======
10 : var3 = 200    <=== [esp]
9 : func2 SFP = 4    <=== [ebp]
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==============================

====== Current Call Stack ======
15 : var4 = 400    <=== [esp]
14 : var3 = 300
13 : func3 SFP = 9    <=== [ebp]
12 : Return Address
11 : arg1 = 77
10 : var3 = 200
9 : func2 SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==============================
```

```
====== Current Call Stack ======
10 : var3 = 200    <=== [esp]
9 : func2 SFP = 4    <=== [ebp]
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

====== Current Call Stack ======
5 : var1 = 100    <=== [esp]
4 : func1 SFP    <=== [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

Stack is empty.
```