call-3.S

```
func:
# put return value into stack location


ret


main:
# grow stack: for return value

call func
# move return value into eax


# reduce stack
```

call-4.S

```
# func should add two args together
# put return value into eax
func:



# grow stack; put 2 args into top
# call func
# reduce stack now; ret val in eax
main:
```

call-5.S

```
# make room for local int
# get two params and add together
# reduce stack (free local int)
func:



# grow stack; put 2 args into top
# call func
# reduce stack
main:
```

call-6.S

```
# save old ebp, set new one
# alloc local int + init to 1
# add params, local var into  eax
# free local vars, restore ebp
func:



# grow stack; put 2 args into top
# call func
# reduce stack
main:
```

call-7.S

```
# save old ebp, set new one
# alloc local int + init to 1
# add params, local var into  eax
# free local vars, restore ebp
func:



# set eax to some value
# save eax (caller save)
# grow stack; put 2 args into top
# call func
# reduce stack
# save retval into ebx
# restore eax
main:
```

assembly-function-1.c

```
int increment(int x) {
    return x + 1;
}

// implement this in assembly
int my_increment(int);

int main() {
    int f = 10;
    printf("value within main %d\n", f);
    f = my_increment(f);
    printf("last value in main %d\n", f);
    return 0;
}
```

assembly-function-2.c

```
void increment(int* x) {
    *x = *x + 1;
}

// implement this in assembly
void my_increment(int*);

int main() {
    int f = 10;
    printf("value within main %d\n", f);
    my_increment(&f);
    printf("last value in main %d\n", f);
    return 0;
}
```