

Let's see a demo of how the stack is built and destroyed during function calls, on a Linux machine using GCC.

Consider this C code

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

Translated to assembly as:

add:

```
    pushl   %ebp  
    movl    %esp, %ebp  
    subl    $16, %esp  
    movl    8(%ebp), %edx  
    movl    12(%ebp), %eax  
    addl    %edx, %eax  
    movl    %eax, -4(%ebp)  
    movl    -4(%ebp), %eax  
    leave  
    ret
```

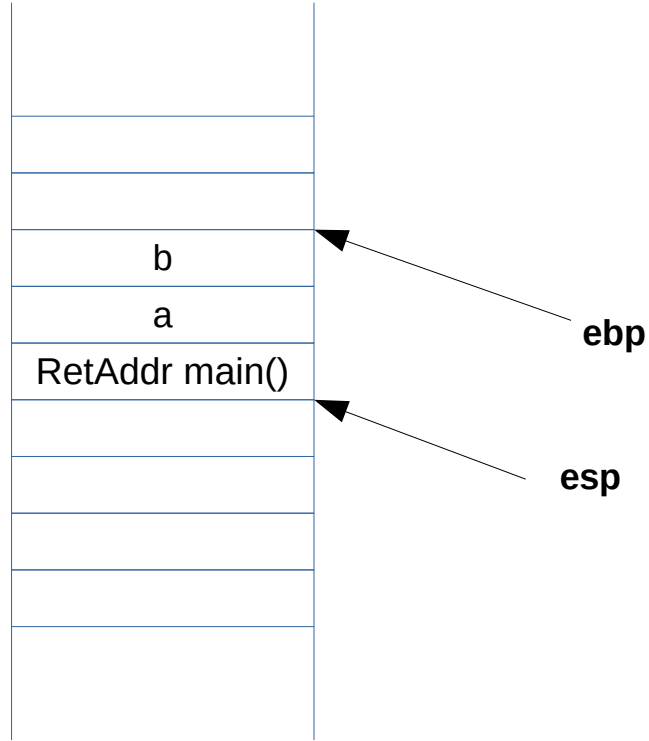
mult:

```
    pushl   %ebp  
    movl    %esp, %ebp  
    subl    $24, %esp  
    movl    $20, -24(%ebp)  
    movl    $30, -20(%ebp)  
    subl    $8, %esp  
    pushl    -20(%ebp)  
    pushl    -24(%ebp)  
    call    add  
    addl    $16, %esp  
    movl    %eax, -16(%ebp)  
    movl    8(%ebp), %eax  
    imull    12(%ebp), %eax  
    movl    %eax, %edx  
    movl    -16(%ebp), %eax  
    addl    %edx, %eax  
    movl    %eax, -12(%ebp)  
    movl    -12(%ebp), %eax  
    leave  
    ret
```

Stack



X



/* Control is here */

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

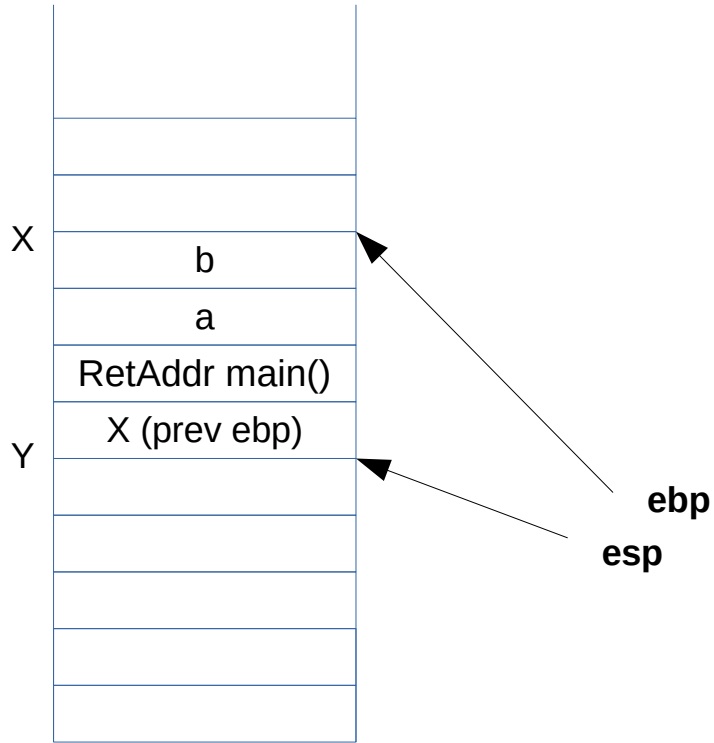


/* Control is here */

mult:

```
pushl    %ebp  
movl     %esp, %ebp  
subl     $24, %esp  
movl     $20, -24(%ebp)  
movl     $30, -20(%ebp)  
subl     $8, %esp  
pushl    -20(%ebp)  
pushl    -24(%ebp)  
call     add
```

Stack

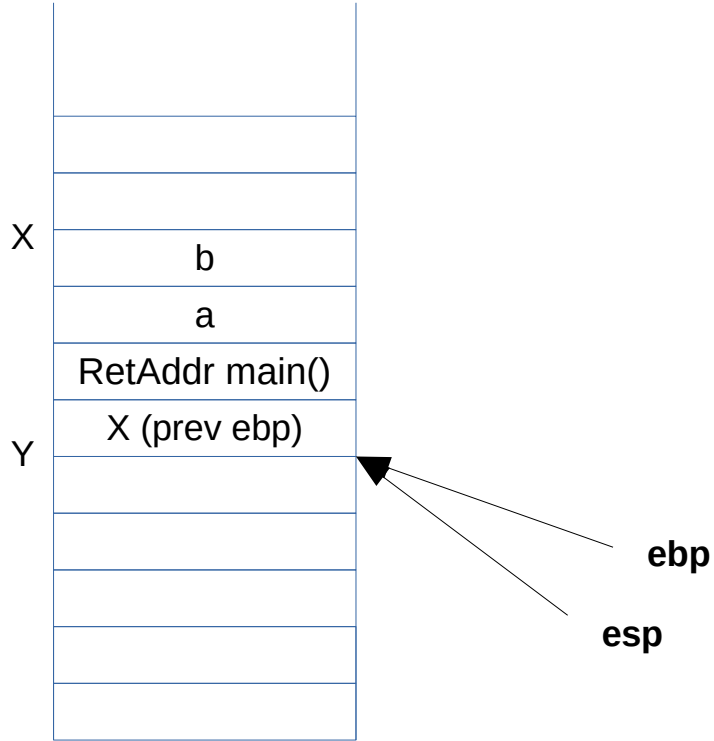


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack

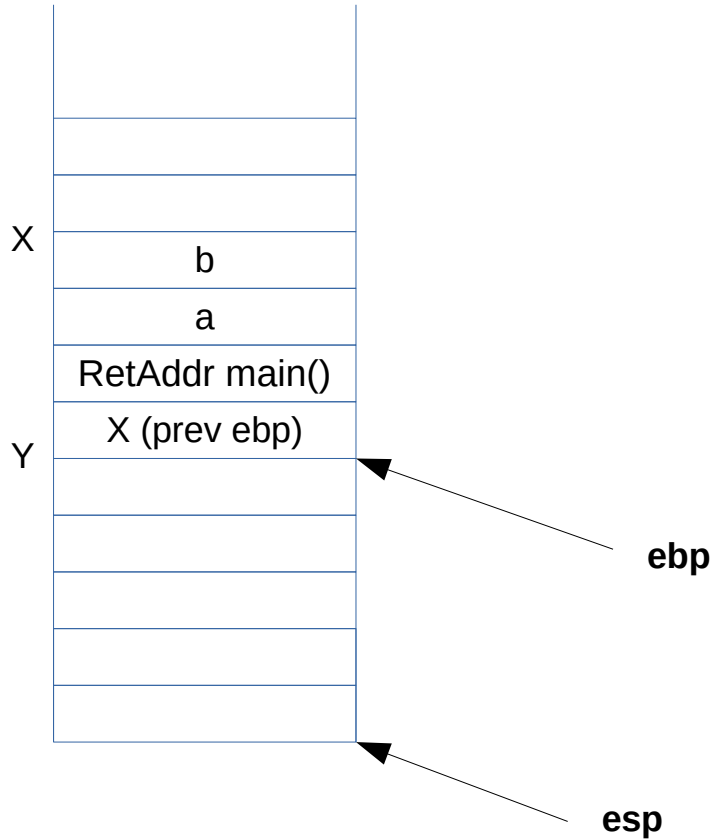


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack

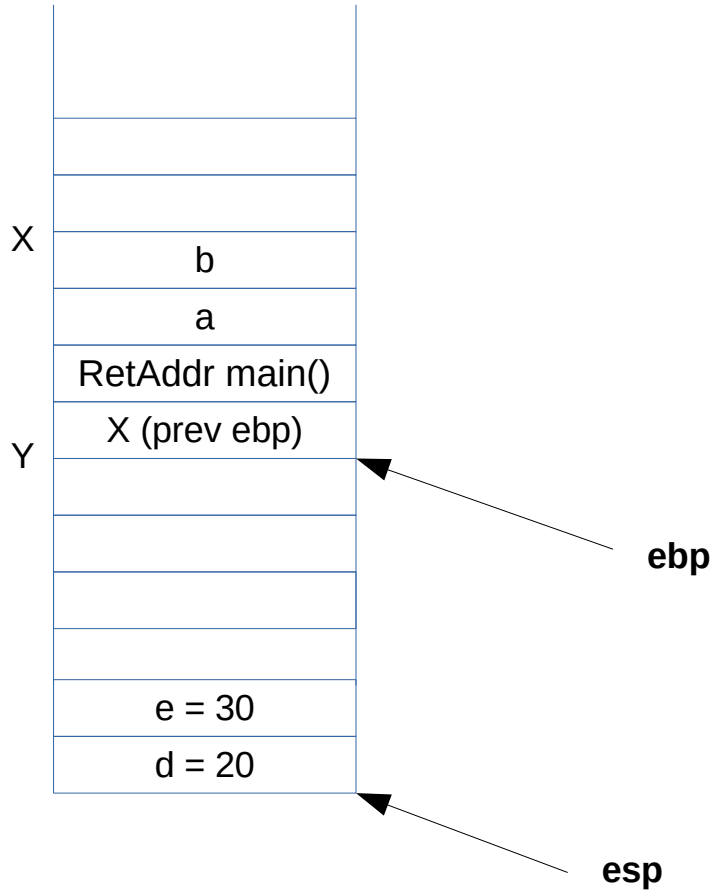


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack

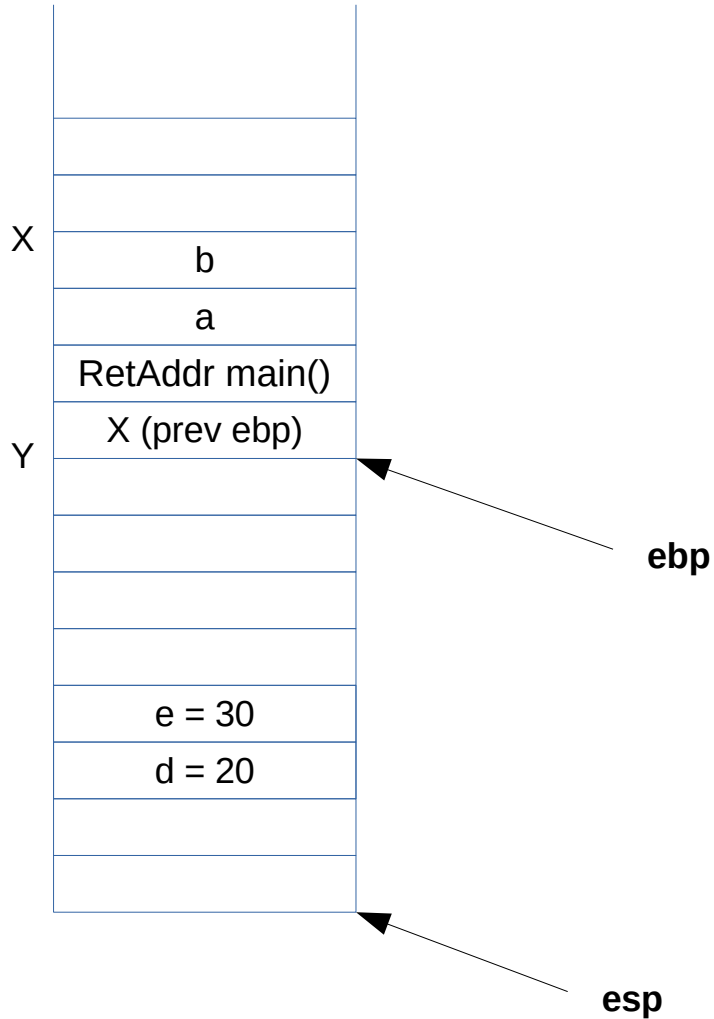


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack

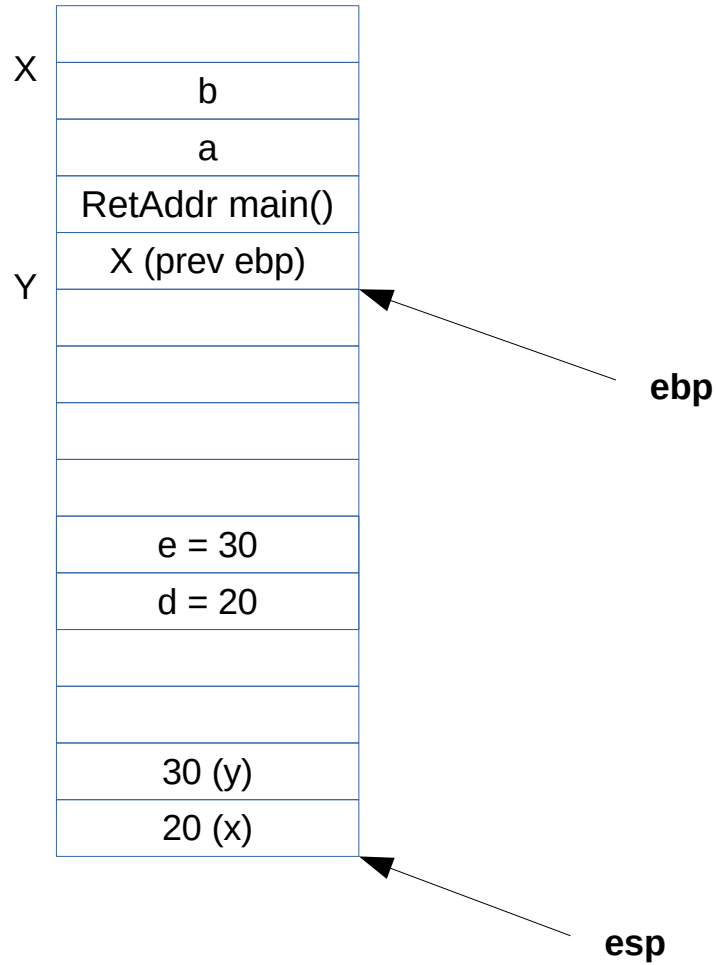


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack

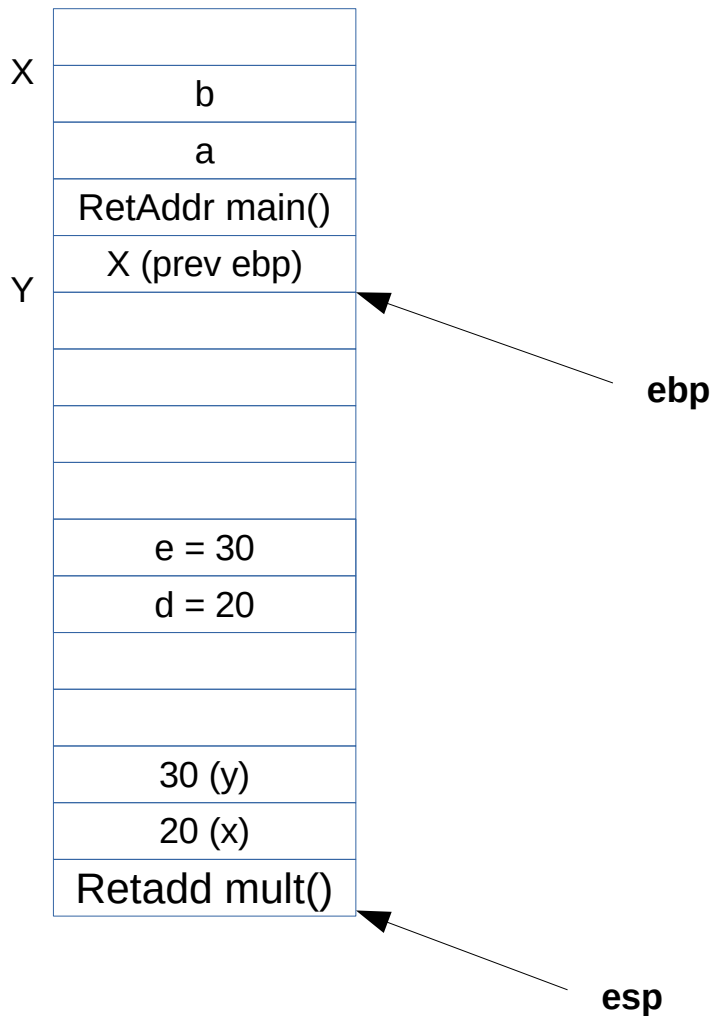


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```


Stack

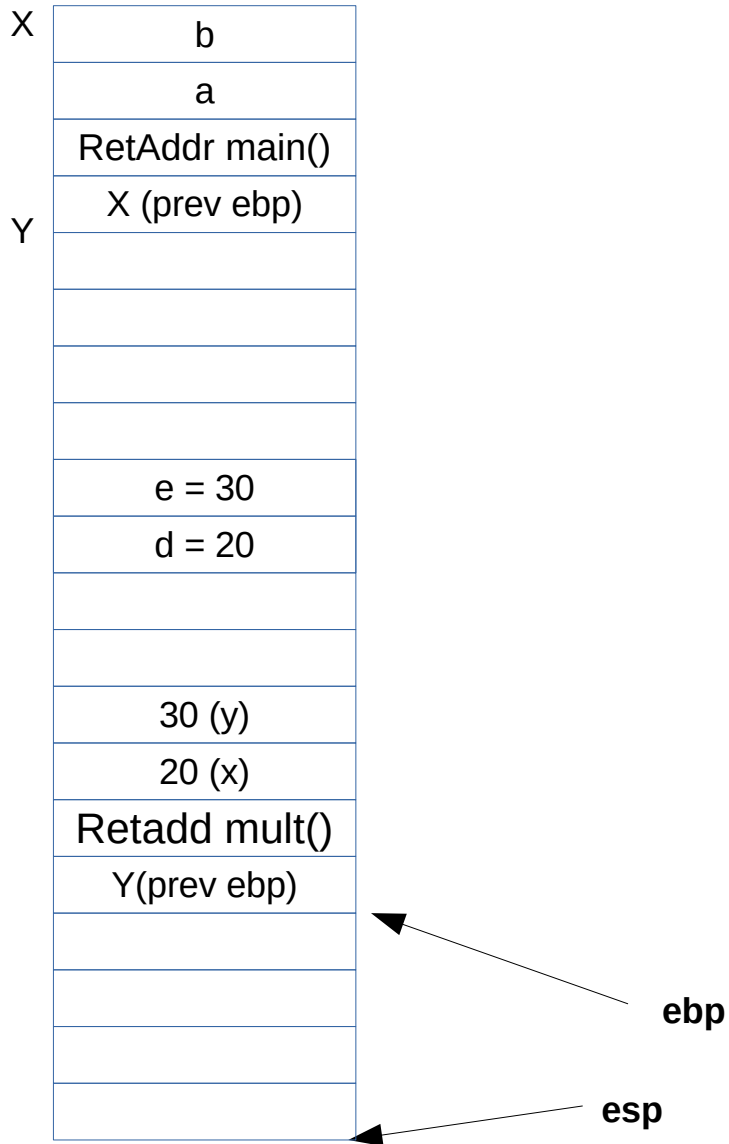


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

mult:

```
pushl %ebp  
movl %esp, %ebp  
subl $24, %esp  
movl $20, -24(%ebp)  
movl $30, -20(%ebp)  
subl $8, %esp  
pushl -20(%ebp)  
pushl -24(%ebp)  
call add
```

Stack



```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

`add:`

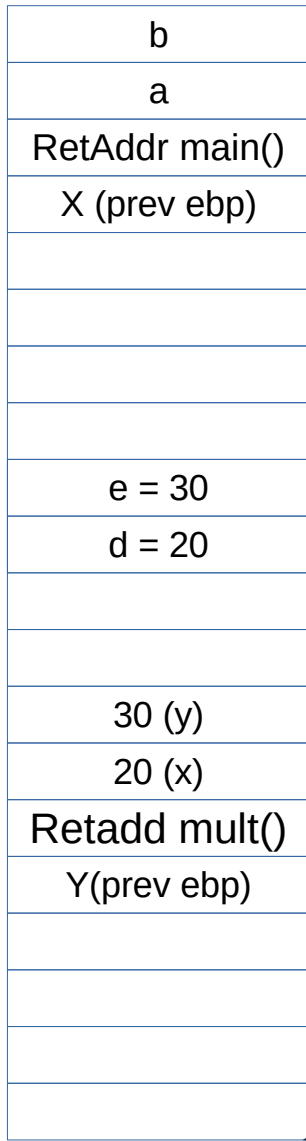
```
pushl    %ebp  
movl     %esp, %ebp  
subl     $16, %esp  
movl     8(%ebp), %edx  
movl     12(%ebp), %eax  
addl     %edx, %eax  
movl     %eax, -4(%ebp)  
movl     -4(%ebp), %eax  
leave  
ret
```

Stack



X

Y



edx = 20
eax = 30
eax = eax + edx = 50

ebp

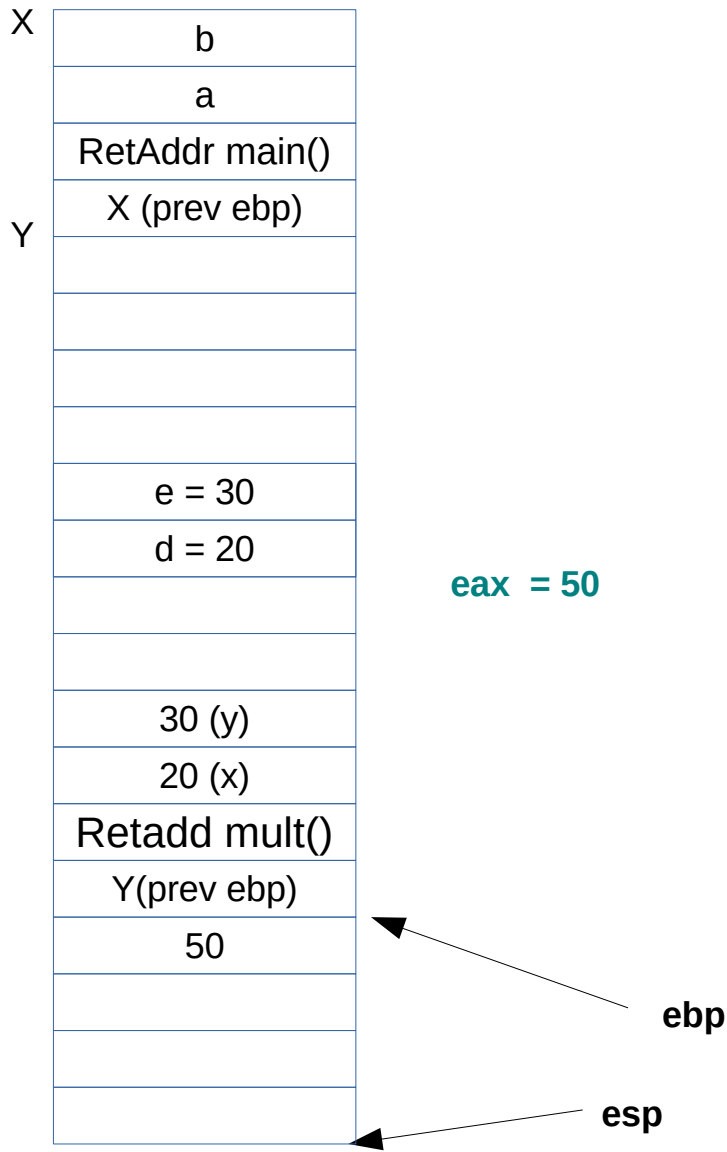
esp

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

add:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
addl %edx, %eax  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax  
leave  
ret
```

Stack

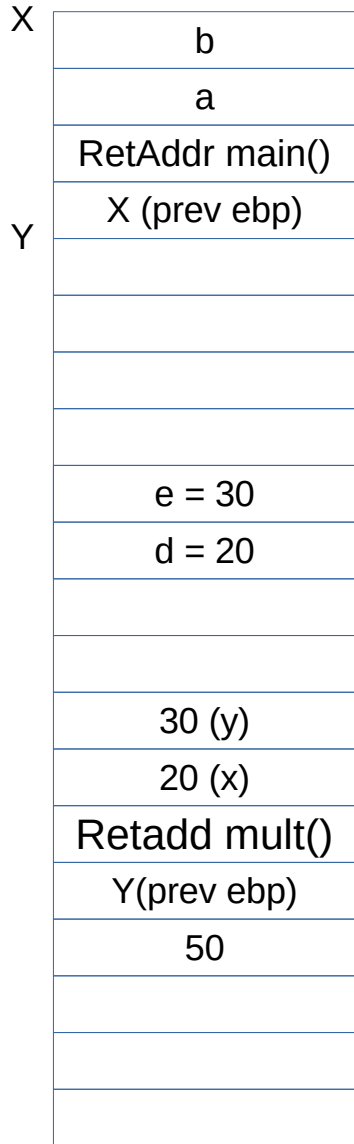


```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

```
add:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $16, %esp  
    movl 8(%ebp), %edx  
    movl 12(%ebp), %eax  
    addl %edx, %eax  
    movl %eax, -4(%ebp)  
    movl -4(%ebp), %eax  
    leave  
    ret
```

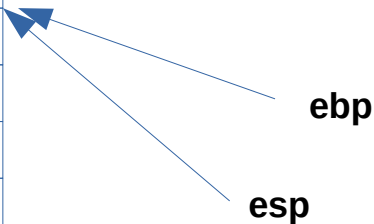
Some redundant code generated here.
Before "leave". Result is in eax

Stack



leave: step 1

eax = 50



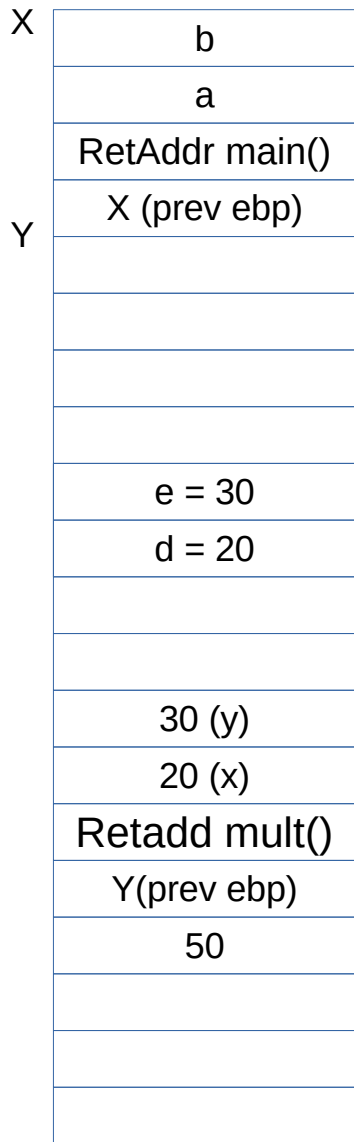
```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

add:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
addl %edx, %eax  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax
```

**leave # # Set ESP to EBP,
then pop EBP.**
ret

Stack



leave: step 2

eax = 50

ebp

esp

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}
```

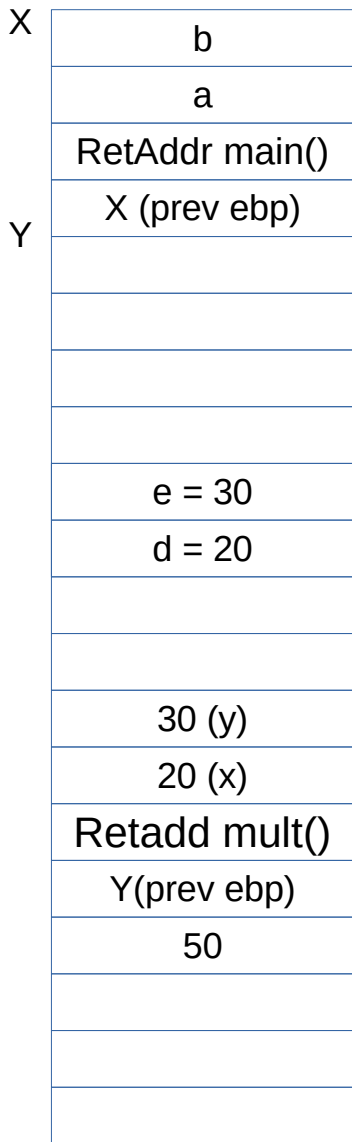
```
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

add:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
addl %edx, %eax  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax
```

leave # # Set ESP to EBP,
then pop EBP.
ret

Stack



eax = 50

ebp

esp

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e); // here  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

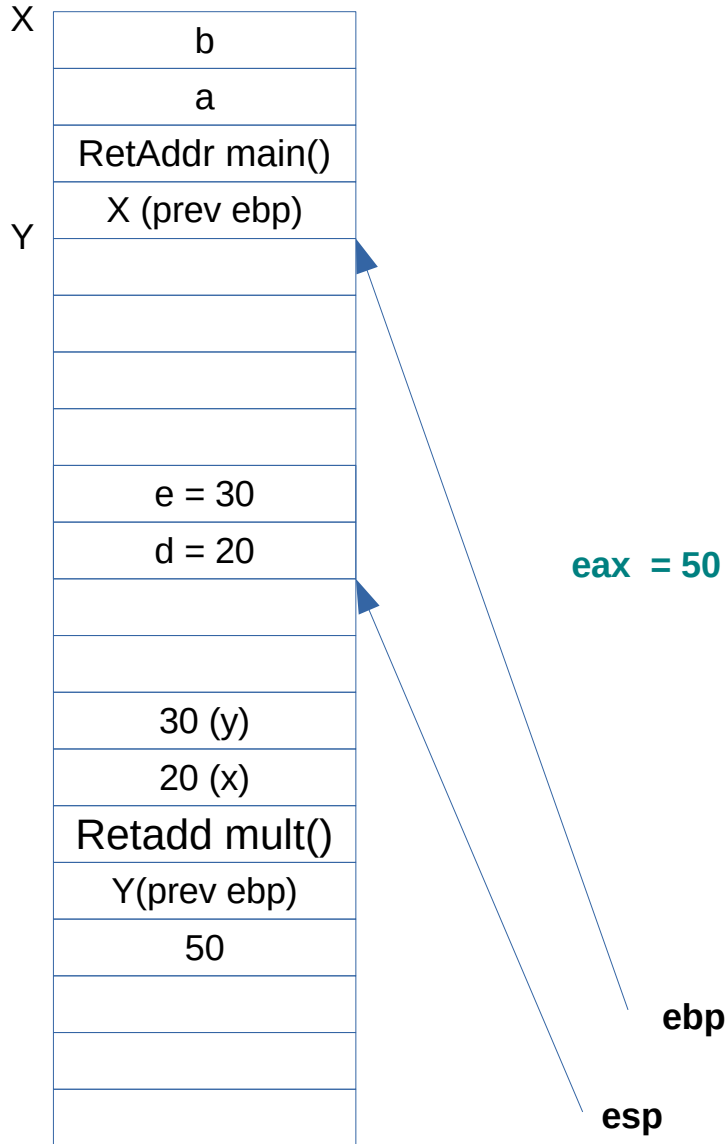
add:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
addl %edx, %eax  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax
```

leave # # Set ESP to EBP,
then pop EBP.

ret

Stack



```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e); // here  
    c = a * b + f;  
    return c;  
}  
  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

Mult:

```
....  
    call    add  
addl    $16, %esp  
    movl    %eax, -16(%ebp)  
    movl    8(%ebp), %eax  
    imull   12(%ebp), %eax  
    movl    %eax, %edx  
    movl    -16(%ebp), %eax  
    addl    %edx, %eax  
    movl    %eax, -12(%ebp)  
    movl    -12(%ebp), %eax  
    leave  
    ret
```


Stack



X

Y

b
a
RetAddr main()
X (prev ebp)
f = 50 (eax)
e = 30
d = 20
30 (y)
20 (x)
Retaddr mult()
Y(prev ebp)
50

eax = 50

ebp

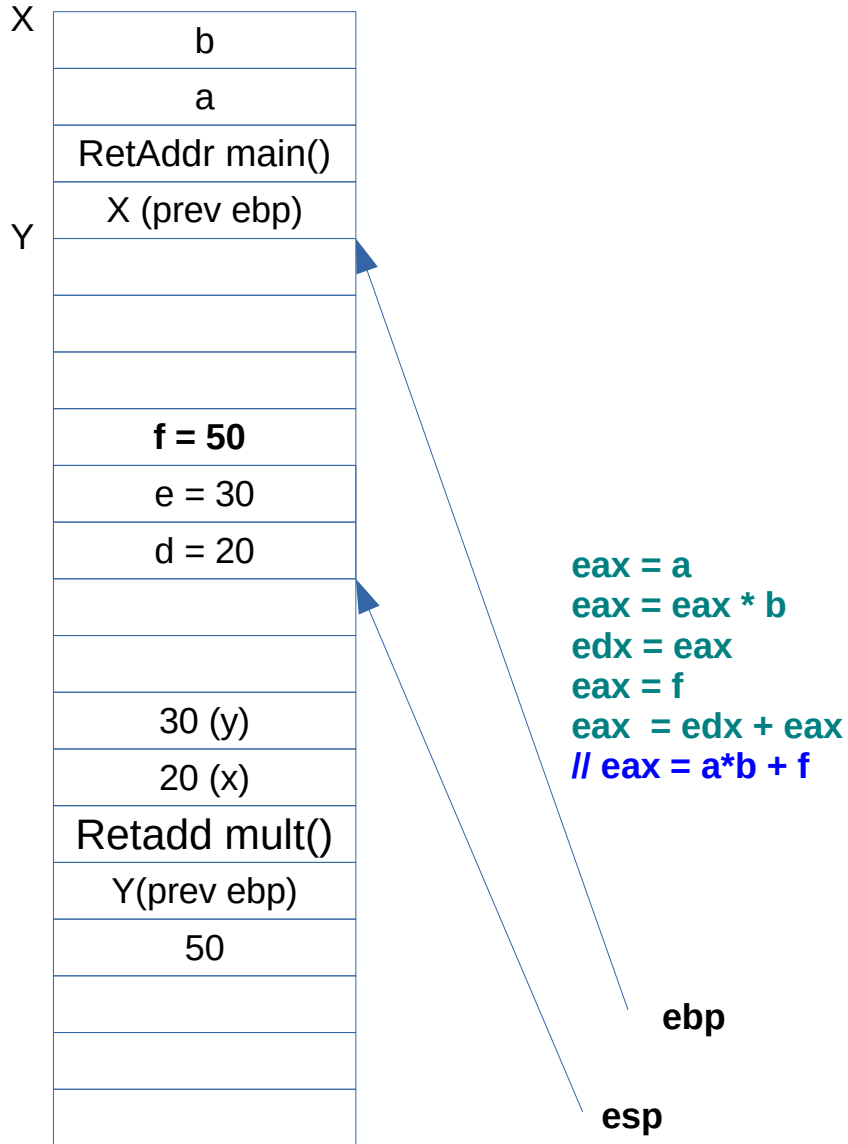
esp

```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}  
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

Mult:

```
....  
    call    add  
    addl    $16, %esp  
    movl    %eax, -16(%ebp)  
    movl    8(%ebp), %eax  
    imull   12(%ebp), %eax  
    movl    %eax, %edx  
    movl    -16(%ebp), %eax  
    addl    %edx, %eax  
    movl    %eax, -12(%ebp)  
    movl    -12(%ebp), %eax  
    leave  
    ret
```

Stack



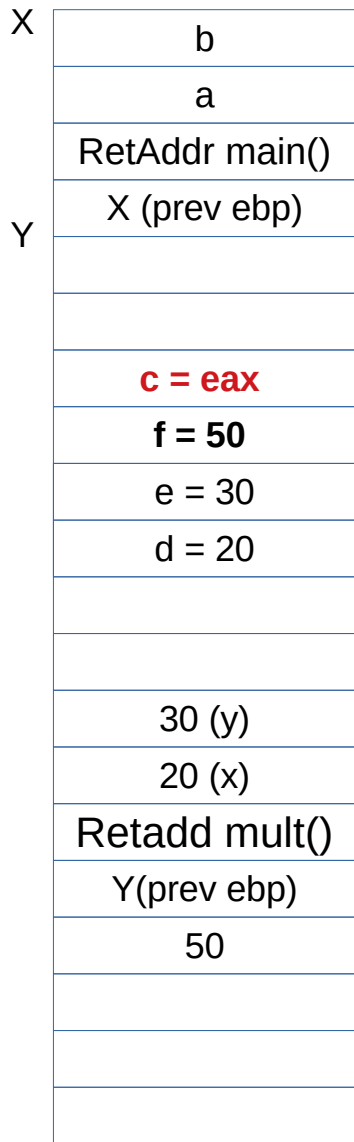
```
int mult(int a, int b) {
    int c, d = 20, e = 30, f;
    f = add(d, e);
    c = a * b + f;
    return c;
}
```

```
int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
```

Mult:

```
....
call    add
addl    $16, %esp
movl    %eax, -16(%ebp)
movl    8(%ebp), %eax
imull   12(%ebp), %eax
movl    %eax, %edx
movl    -16(%ebp), %eax
addl    %edx, %eax
movl    %eax, -12(%ebp)
movl    -12(%ebp), %eax
leave
ret
```

Stack



// eax = a*b + f

Again some
redundant
code

ebp

esp

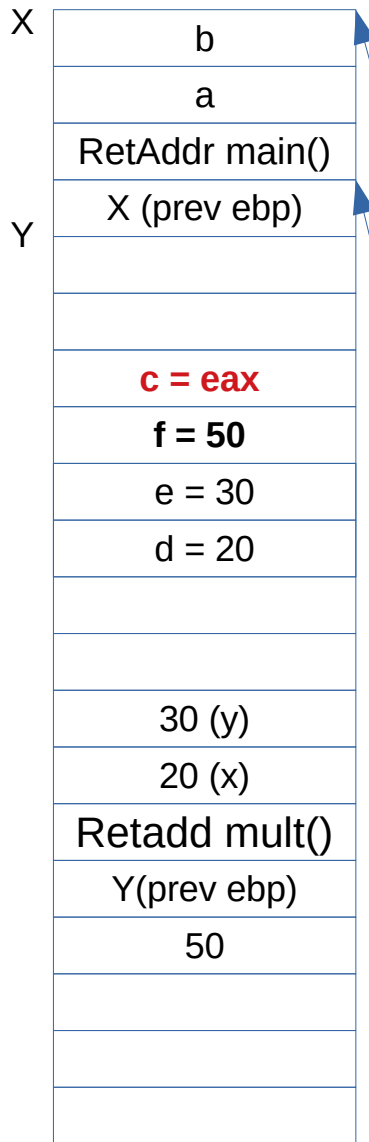
```
int mult(int a, int b) {  
    int c, d = 20, e = 30, f;  
    f = add(d, e);  
    c = a * b + f;  
    return c;  
}
```

```
int add(int x, int y) {  
    int z;  
    z = x + y;  
    return z;  
}
```

Mult:

```
....  
    call    add  
    addl    $16, %esp  
    movl     %eax, -16(%ebp)  
    movl    8(%ebp), %eax  
    imull    12(%ebp), %eax  
    movl    %eax, %edx  
    movl    -16(%ebp), %eax  
    addl    %edx, %eax  
    movl     %eax, -12(%ebp)  
    movl    -12(%ebp), %eax  
    leave  
    ret
```

Stack



After leave
// **eax = a*b + f**

ebp

esp

```
int mult(int a, int b) {
    int c, d = 20, e = 30, f;
    f = add(d, e);
    c = a * b + f;
    return c;
}

int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
```

Mult:

```
....
call    add
addl    $16, %esp
movl    %eax, -16(%ebp)
movl    8(%ebp), %eax
imull   12(%ebp), %eax
movl    %eax, %edx
movl    -16(%ebp), %eax
addl    %edx, %eax
movl    %eax, -12(%ebp)
movl    -12(%ebp), %eax
leave
ret
```

Lessons

- **Calling function (caller)**
 - Pushes arguments on stack , copies values
- **On call**
 - Return IP is pushed
- **Initially in called function (callee)**
 - Old ebp is pushed
 - `ebp = stack`
 - Stack is decremented to make space for local variables

Lessons

- **Before Return**
 - Ensure that result is in 'eax'
- **On Return**
 - `stack = ebp`
 - Pop ebp (`ebp = old ebp`)
- **On 'ret'**
 - Pop 'return IP' and go back in old function

Lessons

- **This was a demonstration for a**
 - User program, compiled with GCC, On Linux
 - Followed the conventions we discussed earlier
- **Applicable to**
 - C programs which work using LIFO function calls
- **Compiler can't generate code using this mechanism for**
 - Functions like `fork()`, `exec()`, `scheduler()`, etc.
 - Boot code of OS