Assembly

- msgHello db "_____", 0
  puts _____

loopStr:
  mov ecx, 100 ; counter
  ✗ CRLF db 10,13,0 ✓ puts CRLF
  ✗ msgHello db "_____", 0 ✓ puts msgHello
  ~~puts msgHello~~
  loop loopStr

- f1(int, short, int) : int UML
  iVal = f1(iVal1, sVal2, iVal3) Java

```
push iVal3
push sVal2
push iVal1
call f1
mov iVal, eax
add esp, 10
```

- sVal = f1(sVal1, sVal2, bVal1) ; must cast

```
mov al, bVal1
cbw                ; ax = bVal1
cwde               ; eax = bVal1
push eax
push sVal2
mov ax, sVal1
cwde
push eax
```

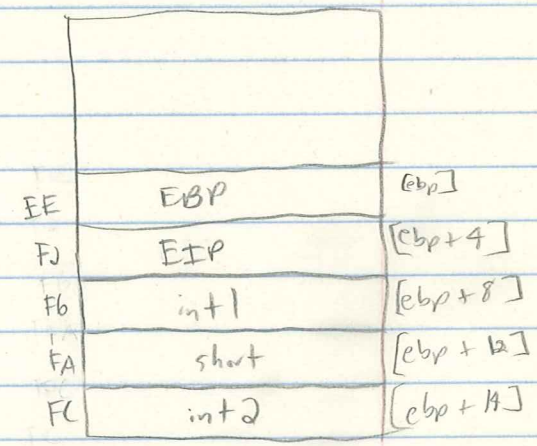
```
call f1
mov sVal, ax
add esp, 10
```

- f1(int, short, int) : int
  iVal = f1(&array1, sVal, &array2)

UML

Java

```
push offset array2
push sVal
push offset array1
call f1
mov iVal, eax
add esp, 10
```



- f2(int, short) : void
  f2(iVal, sVal);

```
push sVal
push iVal
call f2
add esp, 6
```

- f2(int, short) : void
  f2(sVal, bVal)

```
mov al, bVal
cbw                 ; ax = bVal
push ax
mov ax, sVal
cwde                ; eax = sVal
push eax
call f2
add esp, 6
```

- mul - multiply two positive (even if one is negative)
  imul - sign is considered

imul op
can only multiply 1x1, 2x2, or 4x4
;Val = bVal x sVal ; cannot do!
what kind of multiply is determined by operand
op1 is multiplier
;Mul bVal 1 byte multiplier → al, result is in ax
imul sVal 2 byte ... → ax, result is dx:ax
imul dVal 4 byte → eax, result is edx:eax

bVal byte 5
sVal word 10
;Val dword 15

sVal2 = bVal1 x bVal2 ; legal
;Val = sVal1 x sVal2 ; legal
LVal = ;Val1 x ;Val ; legal

- 4 x 5
  mov al, 4
  mov bl, 5
  imul bl ;AX = 20

- 300 x 10
  mov ax, 300
  mov bx, 10
  imul bx ; dx:ax

- iVal = sVal1 x sVal2

```
mov ax, sVal1
imul sVal2          ; dx:ax
mov word ptr iVal, ax
mov word ptr iVal+2, dx
```

or

```
mov ebx, offset iVal
mov [ebx], ax
mov [ebx+2], dx
```

- sVal = sVal1 x sVal2

```
mov ax, sVal
imul sVal2          ; dx:ax
mov sVal, ax        ; dx is lost
```

- LVal = lVal1 x lVal2

```
mov eax, lVal
imul lVal2          ; edx:eax
mov dword ptr LVal, eax
mov dword ptr LVal+4, edx
```

— or —

```
mov ebx, offset LVal
mov ebx, eax
mov ebx+4, edx
```

- byte x byte → anything but sign overflows into AH, CF & OF is set
  word x word → " " dx, CF & OF is set
  dword x dword → " " edx, CF & OF is set

- iVal = sVal x bVal ; convert byte to short

```
mov al, bVal
cbw             ; ax = bVal
imul sVal       ; dx:ax
mov ebx, offset iVal
mov [ebx], ax
mov [ebx+2], dx
```

- iVal = iVal1 x bVal

```
mov al, bVal
cbw             ; ax = bVal
cwde            ; eax = bVal
imul iVal1      ; edx:eax
mov iVal, eax   ; edx is lost
```

- ascint (&string) : int
  intasc (&string, in) : void
  gets
  puts

```
extern ascint:Near32, intasc:Near32
```

for ascint - can only accept numbers
sets carry flag if it's an invalid string
sets overflow flag if it's too big

```
call
jc                      ; use jump carry
jo                      ; use jump overflow
mov :Val, eax
add esp, 4
jmp
```