Assembly

- ascint32

```
push offset strNum
call ascint32
add esp, 4
mov iVal, eax
```

- convasc macro strNum, iVal

```
push eax
push offset strNum
call ascint
add esp, 4
mov iVal, eax
pop eax
endm
```

-
```
iVal1 word ?
iVal2 word ?
strInput1 byte 10 dup(?)
strInput2 byte 10 dup(?)

convasc strInput1, iVal1
convasc strInput2, iVal2
```

- macro must appear somewhere before it is called

```
exitProcess ...
```

MACROS

```
.data
.code
```

- write a putc macro

```
putc    macro    bChar

        push eax
        xor eax, eax;
        mov al bChar
        push eax
        call putchar
        add esp, 4
        pop eax
endm
```

- write a getch macro

```
getch   macro   bChar

        push eax
        xor eax, eax
        call getchar
        mov bChar, al
        pop eax
endm
```

- write a getche macro

```
getche  macro   bChar

        push eax
        xor eax, eax
        call getcharecho
        mov bChar, al
        pop eax
endm
```

- write puts macro

```
.puts macro strMsg

  push eax
  push ecx → mov eax, offset strMsg
  loop:
    mov al, eax + ecx
    cmp al, 0
    je escapeloop
    putc al
    inc ecx
    jmp loop
  escapeloop:
  pop ecx
  pop eax
endm
```

- write code expansion for puts strPrompt1 + puts strPrompt2

the macro expands the labels - local

use:

```
puts macro strMsg
      local loop, escapeLoop
```

- addbytes macro x, y, z

```
push eax
mov al, x
add al, y
add al, z
pop eax
endm
```

- addshorts macro x, y, z

```
push eax
mov ax, x
add ax, y
add ax, z
pop eax
endm
```

- Recursion

$f(0) = 0$

$f(n) = n + f(n-1)$

C code:

```
int f(n)
{
    if(n==0)
        return 0;
    else
        return (n+ f(n));
}
```

```
int f(int n)
{
    if(n==0)
        return 0;
    else
    {
        t=f(n-1);
        v= n+t;
        return v;
    }
}
```

```
y=g(;X);
assembly =      push ;X
                call g
                add esp,4
                mov y, eax
```

```
f proc Near32
push ebp
mov ebp, esp
sub esp, 8          <-------- push ebx
t EQU [ebp-4]
v EQU [ebp-8]
mov eax, 0
cmp dword ptr [ebp+8], n
je done
```

```
    mov ebx, n
    dec ebx
    push ebx
    call f
    add esp, 4
    mov t, eax

    add eax, n
finish:
    pop ebx
    add esp, 8
    pop ebp
    ret

f proc Near32
    push ebp
    mov ebp, esp
    n equ [ebp+8]
    push ebx
    mov eax, 0
    cmp dword ptr n, 0
    je done
    mov ebx, n              ; get to N-1
    dec ebx
    push ebx                ; get ready to call function
    call f
    add esp, 4
    add eax, n
done:
    pop ebx
    pop ebp
    ret
```