

**Q1:**

```
C = 10;
```

```
while(C > 0)
```

```
{
```

```
print(C);
```

```
C = C - 2;
```

```
}
```

output:

```
PS C:\Users\hp\Desktop\labr\q1> C:\masm32\bin\ml /c /coff /Cp prog1.asm
```

```
Microsoft (R) Macro Assembler Version 6.14.8444
```

```
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.
```

```
Assembling: prog1.asm
```

```
PS C:\Users\hp\Desktop\labr\q1> C:\masm32\bin\link -entry:main /subsystem:console prog1.obj
```

```
Microsoft (R) Incremental Linker Version 5.12.8078
```

```
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
```

```
PS C:\Users\hp\Desktop\labr\q1> ./prog1
```

```
10
```

```
8
```

```
6
```

```
4
```

```
2
```

```
PS C:\Users\hp\Desktop\labr\q1> 
```

RAM: 62

Ln 36, Col 15

Spaces: 4

UTF-8

CRLF

Plain Text

Prettier



## Q2:

let X as integer = IN();

let Y as integer = 10 - X;

OUT(Y);

Output:

a)

```
In line no 1, Inserting X with type INT_TYPE in symbol table.  
Parsing finished!
```

```
===== INTERMEDIATE CODE=====
```

```
0: start          -1  
1: ld_int          10  
2: ld_var          0  
3: print_int_value 0  
4: halt           -1
```

```
===== ASM CODE=====
```

b)

```
===== ASM CODE=====
```

```
;start -1
```

```
.686
```

```
.model flat, c
```

```
include C:\masm32\include\msvcrt.inc
```

```
includelib C:\masm32\lib\msvcrt.lib
```

```
.stack 100h
```

```
printf PROTO arg1:Ptr Byte, printlist:VARARG
```

```
scanf PROTO arg2:Ptr Byte, inputlist:VARARG
```

```
.data
```

```
output_integer_msg_format byte "%d", 0Ah, 0
```

```
output_string_msg_format byte "%s", 0Ah, 0
```

input\_integer\_format byte "%d",0

number sdword ?

.code

main proc

```
    push ebp
    mov ebp, esp
    sub ebp, 100
    mov ebx, ebp
    add ebx, 4
```

;ld\_int 10

```
    mov eax, 10
    mov dword ptr [ebx], eax
    add ebx, 4
```

;ld\_var 0

```
    mov eax, [ebp-0]
    mov dword ptr [ebx], eax
    add ebx, 4
```

;print\_int\_value 0

```
    push eax
    push ebx
    push ecx
    push edx
    push [ebp-0]
    push [ebp+4]
    push [ebp+8]
    push ebp
    mov eax, [ebp-0]
    INVOKE printf, ADDR output_integer_msg_format, eax
    pop ebp
    pop [ebp+8]
    pop [ebp+4]
    pop [ebp-0]
```

```
pop edx  
pop ecx  
pop ebx  
pop eax
```

```
;halt -1
```

```
add ebp, 100  
mov esp, ebp  
pop ebp  
ret
```

```
main endp  
end
```