Nested loop

Nested Loops

↓

Loops WithinLoop

①

4 times →

SE1	SE2	SE3	SE4
20	50	40	10
30	60	50	20
40	70	30	30

total 4 loops

Running this

Not good Practice

1) Complexity Increases
2) Chances of Error Increases.
3) length of Program Increase.
4) Wastage of Memory.

Nested loops Used One loop within loop

1 main loop & second loop within it

L1:

L2:
Loop L2 } Nested loop

Loop L1 } Main loop

Need of Nested loop :-

1) Reduce Complexity. (Biggee data) (Only Run for 2 times Main & Nested loop)

2) Maintain Program (Aligned) (We can find error) (Organized).

3).

Use Nested loop

How use it in assembly language.

We have to ruen loop 4 times, so before loop, we write

```
    Mox Cx,4
L1:
   Mov Cx,3
L2:
loop L2
loop L1
```

Nested loop

Main loop.

Set counter Register.

Main loop counter register value is 4 & inested loop CX (counter register) value is 3, when loop starts the value of Main loop replaced by nested loop.
So we push CX & pop CX, With the help of PUSH & POP.

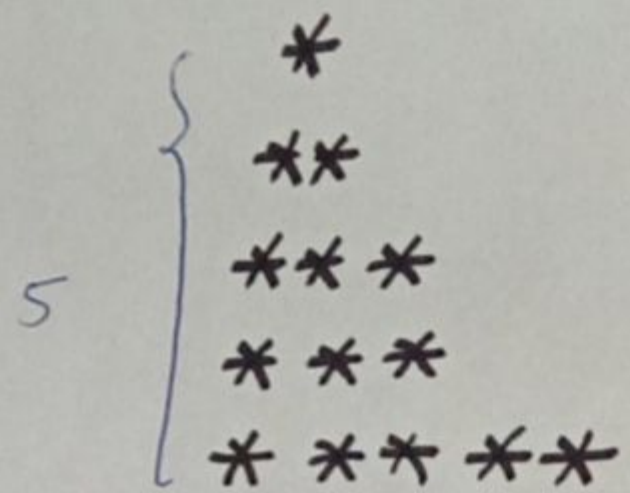
```
Mov cx, 4
L1:
Push cx
Movcx, 3
L2:
loop L2
POP cx
Loop L1
```

Value 4 save in stack first.

then it runs

then we pop it, got back value 4

Pyramid




```
mov cx, 5        ⟶  Main loop runs five times.
L1:
Push cx

L2:
Mov dx, '*'
Mov ah, 2        ⟶  Print *
int 21h
```

```
Mov dx, 10
mov ah, 2
int 21h
Mov dx, 13
mov ah, 2
int 21h
```

Next line

~~enter~~ New line feed
Carriage return

Now, last time we see constant value but this time pyramid pattern
```
*
**
***
```
. so for this how many time to run nested loop, We take register, and put value 1 in it, then we call it & increment it.

```
mov bx, 1
mov cx, 5
L1:
Push cx
Mov cx, bx
L2:
Mov dx, '*'
Mov ah, 2
int 21h
Mov dx, 10
Mov ah, 2
int 21h
```

→ take registe bx & put value 1.

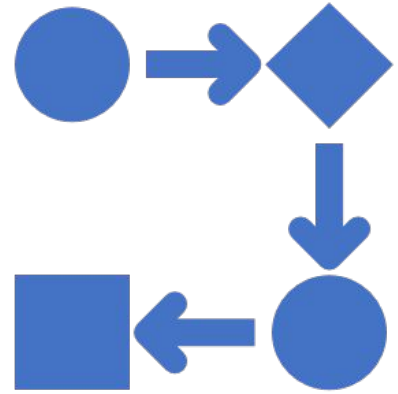
→ loop 1 start

→ value is 1, it run 1 time next time it increment run 2, 3, & as many times.

```
Mov dx, 13
Mov ah, 2
int 21h
Loop L2
Inc bx
Pop cx
Loop L1
```

→ increment in bx, now value is 2.

Procedure



PROCEDURE

If you want to print Any four statements / strings.

{ String 1
String 2
String 3
String 4

After string 1 is printed, for enter you need to write Six lines of codes.

for Enter {
Mov dx, 10
Mov ah, 2
int 21h
Mov dx, 13
Mov ah, 2
int 21h

If print more strings, for every line if we write these six lines, Complexity Increases, Program length increases. Chances of error increases. (not a good practice).

So better to write Code Once, Named it, When needed Call it.

" Procedure : is just a block of code that can be called anywhere in the Program with name"

Need:-

1) Reusability (Used anywhere, anytime we used it)
2) Complexity Reduced (No need to write 6 lines code after every line).

Procedure Use in Assembly

```
Name proc
ret
Name endp
```

Blocked

For enter key Procedure

```
enterkey proc
mov dx,10
mov ah,2
int 21h
Mov dx,13
mov ah,2
int 21h
ret
enterkey endp
```

. How to Use the procedure.
We call it by its name; like → Call Name

String 1
← Call enterkey
String 2

. In assembly program, we write

```
.code
main proc

main endp
```

} It is also a procedure, main is name of our procedure (any name we use but we use main standard way).

Q. No use of ret here, no need of ret only one procedure & when we work on it.

If new procedure made, it call, performed work, ~~performed~~ it also returned.

No need of ret in main.

We follow this pattern

```
.code
main proc

main endp
end main
```

→ One prog end on this

If we make another proc we not write it here, we write it last.

So, code looks like this.

```
.code
main proc
Call enterkey
main endp
enterkey Proc
mov dx, 10
mov ah, 2
int 21h
Mov dx, 13
mov ah, 2
int 21h
Ret
enterkey endp.
End main
```

*. Careful with Names writing.

Initial program structure

```
.data
str1 db 'karachi$'
str2 db 'Lahore$'
str3 db 'Islamabad$'
.code
main proc
mov ax, @data
mov ds, ax
mov dx, offset str1
mov ah, 9
int 21h
call enterkey
mov dx, offset str2
mov ah, 9
int 21h
call enterkey
mov dx, offset str3
mov ah, 9
int 21h
mov ah, 4ch
int 21h
main endp

enterkey proc
mov dx, 10
mov ah, 2
int 21h
mov dx, 13
mov ah, 2
int 21h
ret
enterkey endp
end main
```

Macro

Macro

If want to print strings, for every time the method is,

```
mov dx, offset str1
mov ah, 9
int 21h
```

If multiple strings, same method applied. Best practise is write this once and just give name it will print it.

Q. Procedure can also create for this ?

For Print { mov dx, offset str1
mov ah, 9
int 21h

→

Print proc

Ret
print endp

If we write procedure for it str1 is fixed, Proc not have input again. We have to print multiple strings, and only want to change name of string. Not possible in Procedure.

Macro: is just a block of code that can be used with input parameters anywhere in the program with name.

In Procedure, input not passed, (fixed value), we call multiple times, there we give I/P, then print all seperately. It is perfect function

Like any other language function/method we passed input parameter.

Need: Reusability with Input Parameter
Reduce Complexity.

Macro Used in Assembly

```
Name macro
endm
```

Any parameter to pass write infront of macro (used any name)

```
Name macro p1, p2, ....
Name macro msg1, msg2, ...
```

.data

Str1 ———

Str2 ———

for Print Str1

Print macro P1

endm

Call like

Print Str1

go

(Str1 is P1)

for Print { mov dx, offset P1
mov ah, 9
int 21h

- To called method is, Name P1, P2 → Name(Parameter)
- No need of return in Macro, it is fast.

Difference

Proc	Macro.
No input parameters Ret is used Slow, goes & run code.	Input parameters No ret is used fast, replace with code.

When in prog, create proc, call it and give name of proc,
→ Prog current position move to where proc name & print it, & run code.

In macro print block code is replaced with code,
i.e. why fast, no need of return.

To print string, use macro :-

→ Macro, always created before starting of program.

```
Print macro P1
Mov dx, offset P1
Mov ah, 9
int 21h
endm

. model small
. stack 100h
. data
 Str1 db ———
str2 db ———

. code
main proc
mov ax, @data
mov ds, ax
Print Str1
print Str2

Mov ah, 4ch
int 21h
main endp
end main
```