LAB 10

Advanced Procedures

STUDENT NAME ROLL NO SEC

LAB ENGINEER'S SIGNATURE & DATE

MARKS AWARDED: /

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Prepared by: Qurat ul ain

EL-2003 COMP ORG & ASSEMBLY LANG LAB

LABORATORY MANUAL•FALL 2021 Instructors: Aamir Ali, Aashir Mehboob, Amin Sadiq,Quratulain, M Kariz, Rabia •

Lab Session 10: Advanced Procedures

Learning Objectives

- Implementing procedures using stack frame
- Using stack parameters in procedures
- Passing value type and reference type parameters

Stack Applications

There are several important uses of runtime stacks in programs:

1. A stack makes a convenient temporary save area for registers when they are used for more than one purpose. After they are modified, they *can* be restored to their originalvalues.
2. When the CALL instruction executes, the CPU saves the current subroutine's return address on the stack.
3. When calling a subroutine, you pass input values called arguments by pushing them on the stack.
4. The stack provides temporary storage for local variables inside subroutines.

Stack Parameters

- **Passing by value**
  When an argument is passed by value, a copy of the value is pushed on the stack.

EXAMPLE # 01:

```
.data
var1    DWORD       5
var2    DWORD       6
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
push    ebp
mov     ebp, esp
mov     eax, [ebp + 12]
add     eax, [ebp + 8]
pop     ebp
ret
AddTwo ENDP
```

- **Explicit stack parameters**

  When stack parameters are referenced with expressions such as [ebp+8], we call themexplicit stack parameters.

Example 2:

```
.data
var1    DWORD       5
var2    DWORD       6
y_param         EQU     [ebp + 12]
x_param         EQU     [ebp+ 8]
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
push ebp
mov ebp, esp
mov eax, y_param
add eax, x_param
pop ebp
ret
AddTwo ENDP
```

- **Passing by reference**

  An argument passed by reference consists of the offset of an object to be passed.

EXAMPLE # 03:

```
.data
count = 10
arr     WORD count DUP (?)
.code
push OFFSET arr
push count
call ArrayFill
exit
ArrayFill PROC
push ebp
mov ebp, esp
pushad
```

```
mov esi, [ebp + 12]
mov ecx, [ebp + 8]
cmp ecx, 0
je L2
L1:
mov    eax, 100h
call RandomRange
mov [esi], ax
add esi, TYPE WORD
loop L1
L2:
popad
pop ebp
ret 8
ArrayFill ENDP
```

LEA Instruction

LEA instruction returns the effective address of an indirect operand. Offsets of indirectoperands are calculated at runtime.

EXAMPLE # 04:

```
.code
call      makeArray
exit
makeArray      PROC
push     ebp
mov      ebp, esp
sub      esp, 32
lea      esi, [ebp - 30]
mov ecx,30
L1:
mov      BYTE PTR [esi], '*'
inc      esi
loop     L1
add      esp, 32
pop      ebp ret
makeArray      ENDP
```

ENTER & LEAVE Instructions

Enter instruction automatically creates stack frame for a called Procedure. Leave instructionreverses the effect of enter instruction.

EXAMPLE # 05:

```
.data
var1    DWORD       5
var2    DWORD       6
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
enter 0, 0
mov     eax, [ebp + 12]
add     eax, [ebp + 8]
leave
ret
AddTwo ENDP
```

Local Variables

In MASM Assembly Language, local variables are created at runtime stack, below the basepointer (EBP).

EXAMPLE # 06:

```
.code
call    MySub
exit
MySub PROC
push    ebp
mov     ebp, esp
sub     esp, 8
mov     DWORD       PTR [ebp - 4], 10       ; first parameter
mov     DWORD       PTR [ebp - 8], 20       ; second parameter
mov     esp, ebp
pop     ebp
ret
MySub ENDP
```

LOCAL Directive

LOCAL directive declares one or more local variables by name, assigning them sizeattributes.

EXAMPLE # 07:

```
.code
call LocalProc
```

```
exit
LocalProc PROC
LOCAL  temp : DWORD
mov     temp, 5
mov     eax, temp
ret
LocalProc ENDP
```

Recursive Procedures

Recursive procedures are those that call themselves to perform some task.

EXAMPLE # 08:

```
.code
L1:
mov     ecx, 5
mov     eax, 0
call    CalcSum
call    WriteDec
call    crlf
exit
CalcSum         PROC
cmp     ecx, 0
jz      L2
add     eax, ecx
dec     ecx
call    CalcSum
L2:
ret
CalcSum         ENDP
```

- **INVOKE Directive**

The INVOKE directive pushes arguments on the stack and calls a procedure. INVOKE is a convenient replacement for the CALL instruction because it lets you pass multiple argumentsusing a single line of code.

Here is the general syntax:

```
INVOKE procedureName [, argumentList]
```

For example:

```
push TYPE array

push LENGTHOF array
push OFFSET array

call DumpArray
```

is equal to

```
        INVOKE DumpArray, OFFSET array, LENGTHOF array, TYPE array
```

- **ADDR Operator**

The ADDR operator can be used to pass a pointer argument when calling a procedure usingINVOKE. The following INVOKE statement, for example, passes the address of myArrayto the FillArrayprocedure:

```
        INVOKE FillArray, ADDR myArray
```

- **PROC Directive**

Syntax of the PROC Directive
The PROC directive has the following basic syntax:

```
        Label PROC [attributes] [USES reglist], parameter_list
```

The PROC directive permits you to declare a procedure with a comma-separated list of namedparameters.
Example: The FillArray procedure receives a pointer to an array of bytes:

```
FillArray PROC,
pArray:PTR BYTE
. . .
FillArray ENDP
```

- **PROTO Directive**

The PROTO directive creates a prototype for an existing procedure. A prototype declares a procedure’s name and parameter list. It allows you to call a procedure before defining it and toverify that the number and types of arguments match the procedure definition.

```
        MySub PROTO  ; procedure prototype

        .

        INVOKE MySub  ; procedure call

        .

        MySub PROC              ; procedure implementation

        .

        MySub ENDP
```

Exercises:

1. Write a program which contains a procedure named **BubbleSort** that sorts an array which is passed through a stack using indirect addressing.

2. Write a program which contains a procedure named **TakeInput** which takes input numbers from user and call a procedure named **Armstrong** which checks either a numberis an Armstrong number or not and display the answer on console by calling another function **Display**. (Also show ESP values during nested function calls)

3. Write a program which contains a procedure named **Reverse** that reverse the string using recursion.

4. Write a program which contains a procedure named **LocalSquare** . The procedure mustdeclare a local variable. Initialize this variable by taking an input value from the user andthen display its square. Use **ENTER *& LEAVE*** instructions to allocate and de-allocate the local variable.

5. Write a program that calculates factorial of a given number ***n***. Make a recursive procedure named **Fact** that takes n as an input parameter.

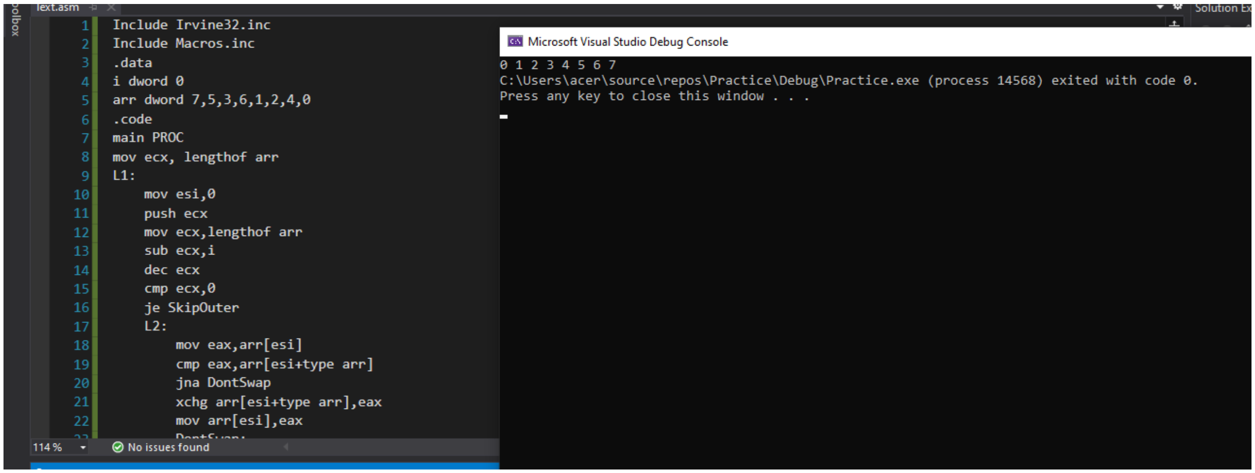
6. Write a program to take 4 input numbers from the users. Then make two procedures **CheckPrime** and **LargestPrime**. The program should first check if a given number is a prime number or not. If all of the input numbers are prime numbers then the program should call the procedure LargestPrime.

CheckPrime: This procedure tests if a number is prime or not

LargestPrime: This procedure finds and displays the largest of the four prime numbers.

LAB 10

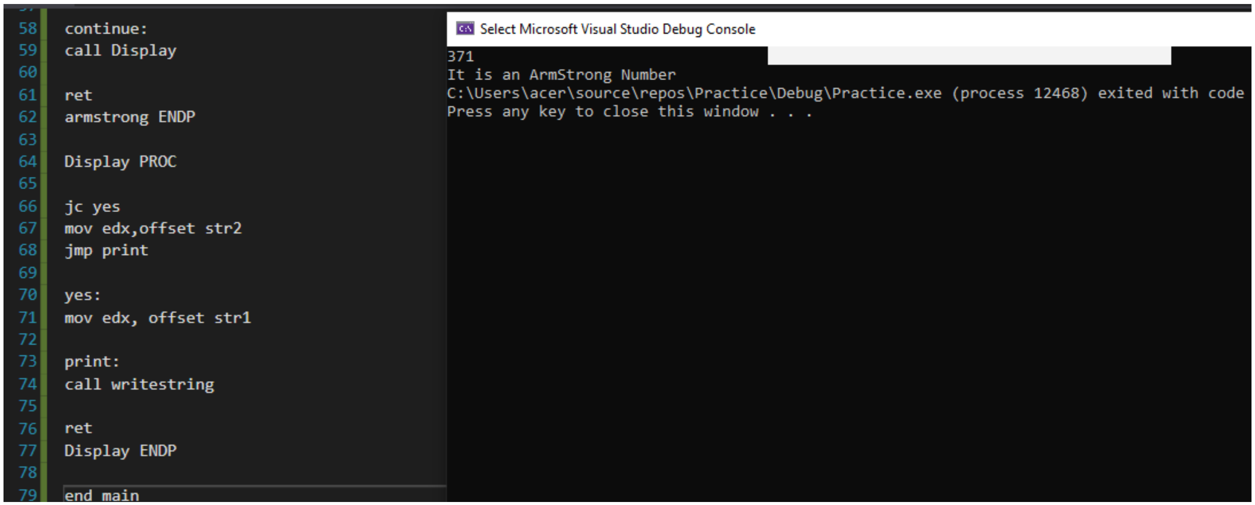
Q1 Code + Output:



```
Include Irvine32.inc
Include Macros.inc
.data
i dword 0
arr dword 7,5,3,6,1,2,4,0
.code
main PROC
mov ecx, lengthof arr
L1:
    mov esi,0
    push ecx
    mov ecx,lengthof arr
    sub ecx,i
    dec ecx
    cmp ecx,0
    je SkipOuter
    L2:
        mov eax,arr[esi]
        cmp eax,arr[esi+type arr]
        jna DontSwap
        xchg arr[esi+type arr],eax
        mov arr[esi],eax
        DontSwap:
```

```
0 1 2 3 4 5 6 7
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 14568) exited with code 0.
Press any key to close this window . . .
```

Q2 Code + Output:

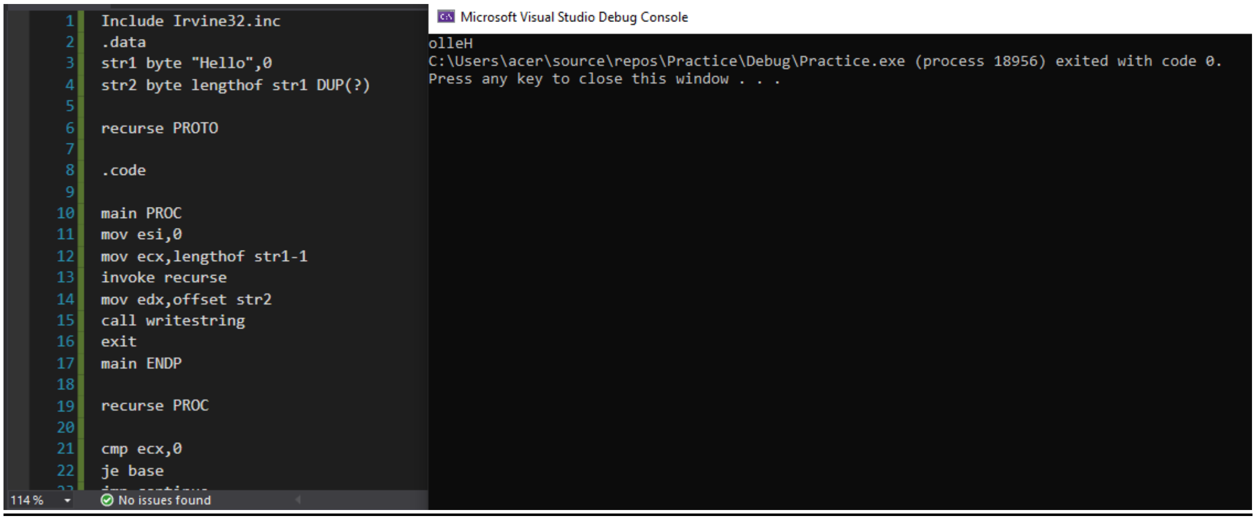


```
continue:
call Display

ret
armstrong ENDP

Display PROC

jc yes
mov edx,offset str2
jmp print

yes:
mov edx, offset str1

print:
call writestring

ret
Display ENDP

end main
```

```
371
It is an ArmStrong Number
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 12468) exited with code
Press any key to close this window . . .
```

Q3 Code + Output:

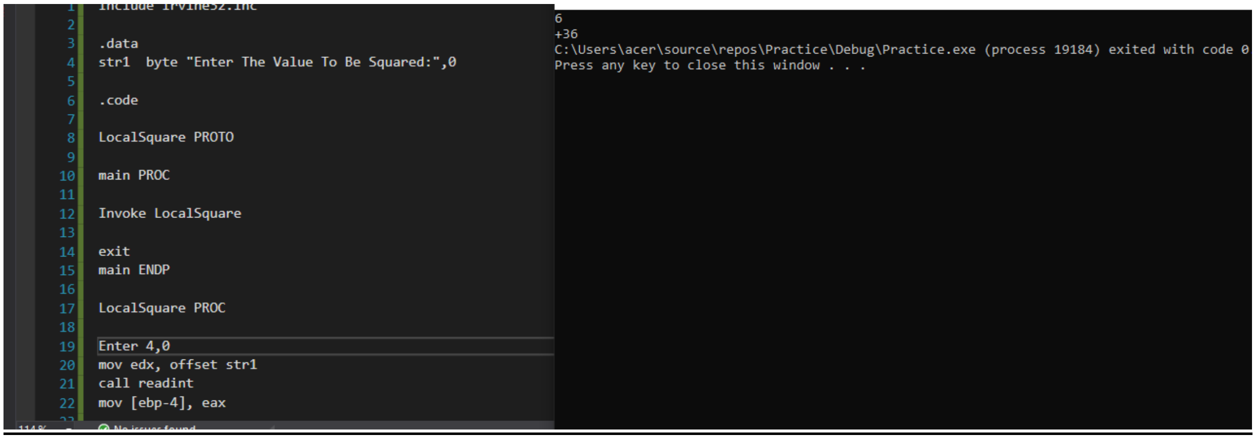
LAB 10



```
Include Irvine32.inc
.data
str1 byte "Hello",0
str2 byte lengthof str1 DUP(?)

recurse PROTO

.code

main PROC
mov esi,0
mov ecx,lengthof str1-1
invoke recurse
mov edx,offset str2
call writestring
exit
main ENDP

recurse PROC

cmp ecx,0
je base
```

```
olleH
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 18956) exited with code 0.
Press any key to close this window . . .
```

Q4 Code + Output:

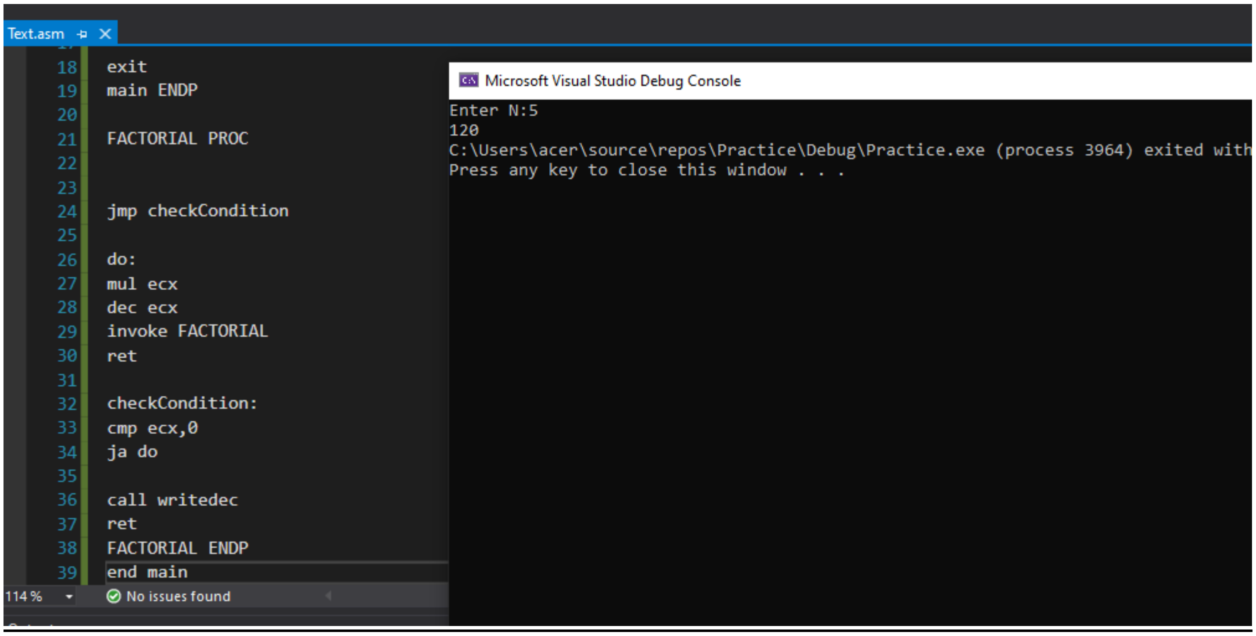


```
Include Irvine32.inc

.data
str1  byte "Enter The Value To Be Squared:",0

.code

LocalSquare PROTO

main PROC

Invoke LocalSquare

exit
main ENDP

LocalSquare PROC

Enter 4,0
mov edx, offset str1
call readint
mov [ebp-4], eax
```

```
6
+36
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 19184) exited with code 0
Press any key to close this window . . .
```

Q5 Code + Output:

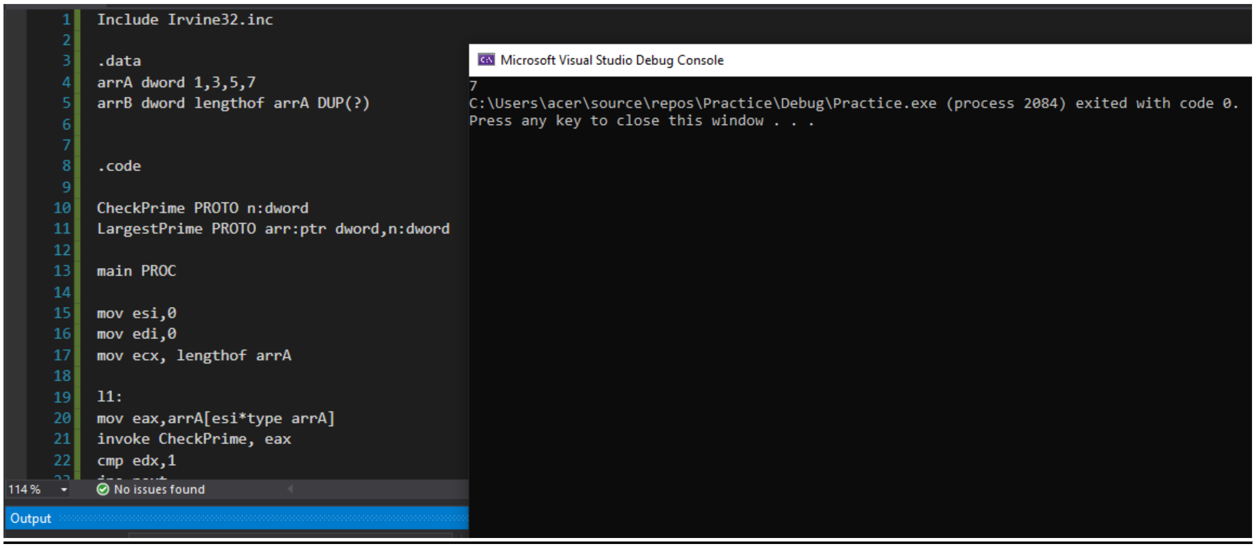
LAB 10



```
exit
main ENDP

FACTORIAL PROC


jmp checkCondition

do:
mul ecx
dec ecx
invoke FACTORIAL
ret

checkCondition:
cmp ecx,0
ja do

call writedec
ret
FACTORIAL ENDP
end main
```

```
Enter N:5
120
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 3964) exited with
Press any key to close this window . . .
```

Q6 Code + Output:

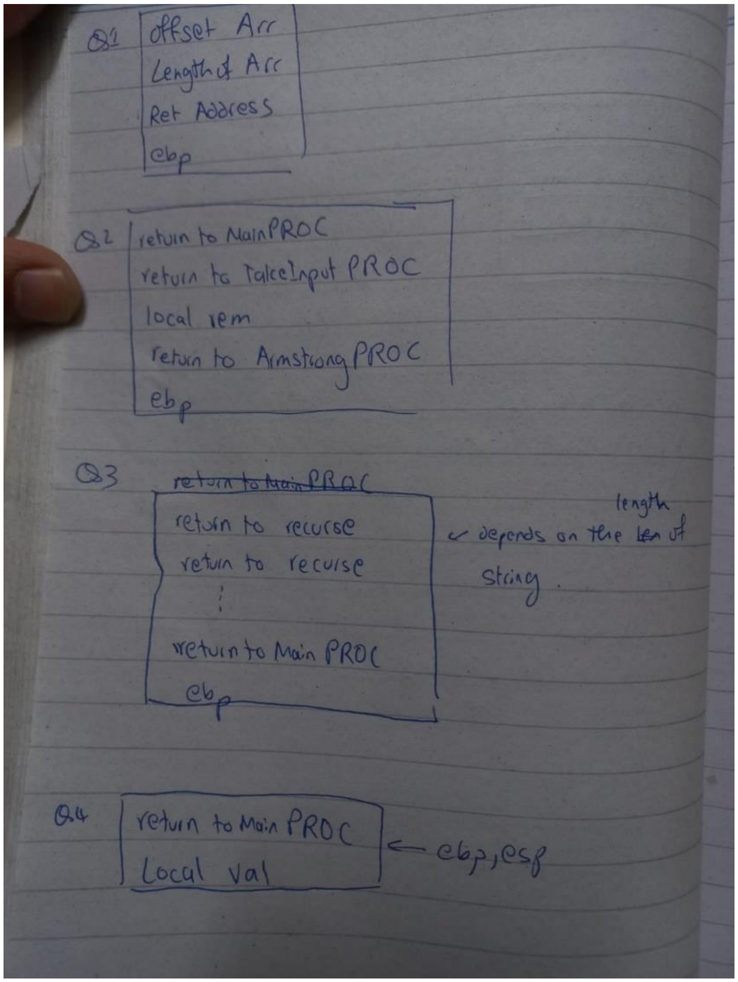


```
Include Irvine32.inc

.data
arrA dword 1,3,5,7
arrB dword lengthof arrA DUP(?)


.code

CheckPrime PROTO n:dword
LargestPrime PROTO arr:ptr dword,n:dword

main PROC

mov esi,0
mov edi,0
mov ecx, lengthof arrA

l1:
mov eax,arrA[esi*type arrA]
invoke CheckPrime, eax
cmp edx,1
jne next
```

```
7
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 2084) exited with code 0.
Press any key to close this window . . .
```

STACK FRAMES:

LAB 10



Q1

Offset Arr
Length of Arr
Ret Address
ebp

Q2

return to MainPROC
return to TakeInput PROC
local rem
return to ArmstrongPROC
ebp

Q3 ~~return to Main~~PROC

return to recurse
return to recurse
⋮
return to Main PROC
ebp

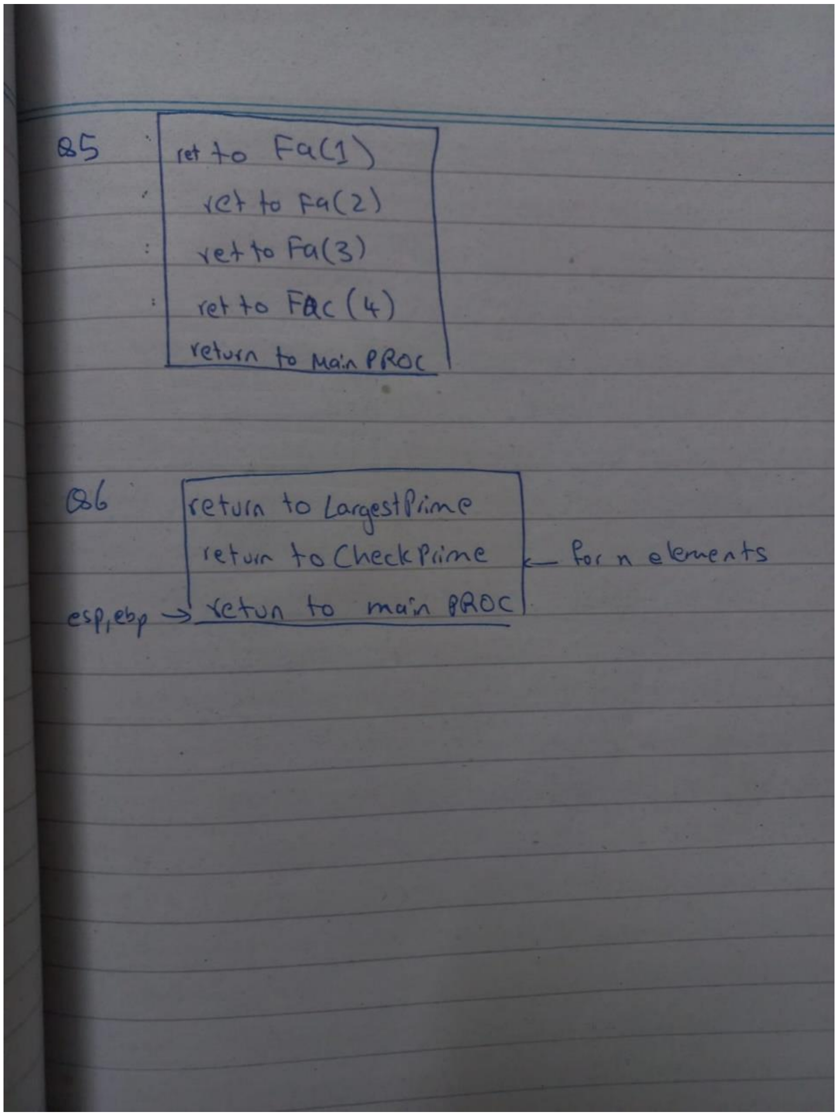
← depends on the ~~len~~ length of string.

Q4

return to Main PROC
Local val

← ebp, esp

LAB 10



Q5

ret to Fa(1)

ret to Fa(2)

ret to Fa(3)

ret to Fac (4)

return to Main PROC

Q6

return to LargestPrime

return to CheckPrime ← for n elements

esp,ebp → retun to main PROC