Experiment 5

Code:

```
.MODEL SMALL

.STACK 100H


.DATA

   prompt1     db 'Enter a string (max 20 chars): $'

   prompt2     db 0Dh,0Ah,'Enter character to search: $'

   foundMsg    db 0Dh,0Ah,'Character found in string.$'

   notFoundMsg db 0Dh,0Ah,'Character NOT found in string.$'


   str1        db 21, 0, 21 dup('$')  ; input buffer (max 20 chars + CR)

   str2        db 21 dup(?)           ; destination buffer for copy

   charToFind  db ?                   ; character to search


.CODE

MAIN PROC

   MOV AX, @DATA

   MOV DS, AX

   MOV ES, AX        ; ES used for SCASB, point it to DS


   ; Prompt for string

   LEA DX, prompt1

   MOV AH, 09H

   INT 21H


   ; Read string into str1 using function 0Ah (buffered input)

   LEA DX, str1

   MOV AH, 0Ah
```

```
INT 21H

; Get string length from str1+1
MOV CL, str1+1
XOR CH, CH        ; CX = length
MOV BX, CX        ; Save for later

; Copy string from str1+2 to str2 using REP MOVSB
LEA SI, str1+2
LEA DI, str2
CLD
REP MOVSB

; Prompt for character to search
LEA DX, prompt2
MOV AH, 09H
INT 21H

; Get one character from user
MOV AH, 01H
INT 21H
MOV charToFind, AL

; Prepare for SCASB to search character
MOV AL, charToFind
LEA DI, str2
MOV CX, BX        ; CX = string length
CLD
REPNZ SCASB
```

```
    ; Check result
    JZ FOUND

    ; Not found
    LEA DX, notFoundMsg
    MOV AH, 09H
    INT 21H
    JMP EXIT

FOUND:
    LEA DX, foundMsg
    MOV AH, 09H
    INT 21H

EXIT:
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

Explanation:

.MODEL SMALL

- Assembler directive: selects the *small* memory model (one 64K code segment, one 64K data segment). Not executed at runtime.

.STACK 100H

- Reserve 0x100 (256) bytes for the stack. Not executed at runtime.

.DATA

- Start of data segment.

prompt1 db 'Enter a string (max 20 chars): $'

- A dollar-terminated string used with DOS INT 21h AH=09 (print string).

prompt2 db 0Dh,0Ah,'Enter character to search: $'

- Another print string preceded by CR+LF (new line), also terminated by $.

foundMsg db 0Dh,0Ah,'Character found in string.$'
notFoundMsg db 0Dh,0Ah,'Character NOT found in string.$'

- Messages for found / not-found cases. Both are $-terminated for INT 21h AH=09.

str1 db 21, 0, 21 dup('$') ; input buffer (max 20 chars + CR)

- Input buffer for INT 21h AH=0Ah (buffered input). Structure:
    - str1[0] = 21 → maximum number of characters the buffer can hold (decimal 21).
    - str1[1] = 0 → will be set by DOS to the number of characters actually entered.
    - str1[2..] → storage area for the typed characters (initialized with $ here).
    - After function 0Ah, the actual characters are at str1+2, and str1+1 holds the count.

str2 db 21 dup(?) ; destination buffer for copy

- Destination buffer (uninitialized) where we will copy the input string. Same size as the data area.

charToFind db ? ; character to search

- One byte storage to hold the character the user wants to search for.

.CODE

- Start of code segment.

MAIN PROC

- Procedure start (entry point label).

MOV AX, @DATA

- Load AX with the segment address of the DATA segment (assembler-provided @DATA).

MOV DS, AX

- Set DS to point to the data segment (so data references use DS).

MOV ES, AX ; ES used for SCASB, point it to DS

- Set ES = DS. SCASB string instruction uses ES:DI as destination to search, so ES must point to the data segment.

```
; Prompt for string
LEA DX, prompt1
MOV AH, 09H
INT 21H
```

- Print prompt1 using DOS INT 21h AH=09. LEA DX, prompt1 loads the offset of the string into DX. DOS prints from DS:DX up to the $ terminator.

```
; Read string into str1 using function 0Ah (buffered input)
LEA DX, str1
MOV AH, 0Ah
INT 21H
```

- Buffered input (DOS INT 21h AH=0Ah). DX points to the buffer str1. After return: str1+1 = count of characters typed, and str1+2.. contain the characters typed (no $ required). The function echoes as typed and stops on Enter.

```
; Get string length from str1+1
MOV CL, str1+1
XOR CH, CH ; CX = length
MOV BX, CX ; Save for later
```

- MOV CL, str1+1 loads the byte at str1+1 (the count) into CL. XOR CH,CH clears CH so CX now contains the length (0..21).
- MOV BX,CX saves this length in BX. This is necessary because the code will use CX for string operations (and those operations will change CX).

Note: assembler syntax may require MOV CL, BYTE PTR [str1+1] — but intent is: load the length byte from the input buffer into CX.

```
; Copy string from str1+2 to str2 using REP MOVSB
LEA SI, str1+2
LEA DI, str2
CLD
REP MOVSB
```

- Prepare to copy the characters:

    - SI = address of source (str1+2 where characters start), DI = address of destination (str2).
    - CLD clears the Direction Flag so string instructions increment SI and DI (forward copy).
    - REP MOVSB copies CX bytes from [DS:SI] → [ES:DI], decrementing CX each time. After this, CX becomes 0 and SI/DI are advanced past the copied bytes.
- Because REP MOVSB consumes CX, that's why we earlier saved the original length in BX.

```
; Prompt for character to search
LEA DX, prompt2
MOV AH, 09H
INT 21H
```

- Print the prompt asking the user to enter the character to search (uses AH=09 again).

```
; Get one character from user
MOV AH, 01H
INT 21H
MOV charToFind, AL
```

- DOS INT 21h AH=01 reads a single character from stdin, echoes it, and returns the ASCII code in AL. That byte is stored into charToFind.

```
; Prepare for SCASB to search character
MOV AL, charToFind
LEA DI, str2
MOV CX, BX ; CX = string length
CLD
REPNZ SCASB
```

- MOV AL, charToFind — load AL with the target character to compare against.
- LEA DI, str2 — point DI to the start of str2 (the copied string). Note: we re-LEA DI because earlier REP MOVSB advanced DI to the end.
- MOV CX, BX — restore CX to the saved length (so SCASB will examine that many bytes).
- CLD — ensure forward scanning.
- REPNZ SCASB — this repeats SCASB while ZF = 0 and CX ≠ 0. SCASB compares AL to byte at ES:DI, sets flags accordingly and increments DI. REPNZ SCASB stops either

when a match is found (ZF=1) or when CX reaches zero (end-of-buffer). After the instruction:

- If a match was found, ZF = 1 and CX > 0, DI points one past the matching byte.
- If not found, CX = 0 and ZF = 0.

```
; Check result
JZ FOUND
```

- JZ jumps if Zero Flag = 1. So if REPNZ SCASB found a match, ZF==1 and we jump to FOUND. Otherwise execution falls through to "not found" handling.

```
; Not found
LEA DX, notFoundMsg
MOV AH, 09H
INT 21H
JMP EXIT
```

- Print the NOT FOUND message using AH=09, then jump to EXIT.

```
FOUND:
LEA DX, foundMsg
MOV AH, 09H
INT 21H
```

- If we jumped here, print the FOUND message.

```
EXIT:
MOV AH, 4CH
INT 21H
```

- DOS terminate program / return to DOS (INT 21h AH=4Ch). AL may contain an exit code (not set here, so whatever is in AL is returned).

```
MAIN ENDP
END MAIN
```