String Library Demo Program

The "String Library Demo" program demonstrates the usage of string-handling procedures from the Irvine32 library. It performs the following tasks:

Trimming trailing characters from string_1 using the Str_trim procedure.

Converting string_1 to uppercase using the Str_ucase procedure.

Comparing string_1 to string_2 using the Str_compare procedure.

Displaying the length of string_2 using the Str_length procedure.
Here's the code with detailed explanations:

```
INCLUDE Irvine32.inc
.data
string_1 BYTE "abcde////", 0
string_2 BYTE "ABCDE", 0
msg0 BYTE "string_1 in upper case: ", 0
msg1 BYTE "string_1 and string_2 are equal", 0
msg2 BYTE "string_1 is less than string_2", 0
msg3 BYTE "string_2 is less than string_1", 0
msg4 BYTE "Length of string_2 is ", 0
msg5 BYTE "string_1 after trimming: ", 0

.code
main PROC
    call trim_string   ; Remove trailing characters from string_1.
    call upper_case    ; Convert string_1 to uppercase.
    call compare_strings ; Compare string_1 to string_2.
    call print_length  ; Display the length of string_2.
    exit
main ENDP

trim_string PROC
    ; Remove trailing characters from string_1.
    INVOKE Str_trim, ADDR string_1, '/'
    mov edx, OFFSET msg5
    call WriteString
    mov edx, OFFSET string_1
    call WriteString
    call Crlf
    ret
trim_string ENDP
```

```
upper_case PROC
    ; Convert string_1 to upper case.
    mov edx, OFFSET msg0
    call WriteString
    INVOKE Str_ucase, ADDR string_1
    mov edx, OFFSET string_1
    call WriteString
    call Crlf
    ret
upper_case ENDP

compare_strings PROC
    ; Compare string_1 to string_2.
    INVOKE Str_compare, ADDR string_1, ADDR string_2
    .IF ZERO?
        mov edx, OFFSET msg1
    .ELSEIF CARRY?
        mov edx, OFFSET msg2
    .ELSE
        mov edx, OFFSET msg3
    .ENDIF
    call WriteString
    call Crlf
    ret
compare_strings ENDP

print_length PROC
    ; Display the length of string_2.
    mov edx, OFFSET msg4
    call WriteString
    INVOKE Str_length, ADDR string_2
    call WriteDec
    call Crlf
    ret
print_length ENDP

END main
```

The program's output is as follows:
After trimming string_1, it displays "string_1 after trimming: abcde."

After converting string_1 to uppercase, it displays "string_1 in upper case: ABCDE."

It then compares string_1 and string_2 and displays one of the messages depending on the result.

Finally, it displays the length of string_2.

This program showcases the use of various string-handling procedures from the Irvine32 library and provides informative messages for each step.

Strings using Irvine64

```
INCLUDE Irvine64.inc
.data
    source BYTE "AABCDEFGAABCDFG", 0
    ; size = 15
    target BYTE 20 DUP(0)

.code
    Str_compare PROTO
    Str_length PROTO
    Str_copy PROTO
    ExitProcess PROTO

    main PROC
        mov rcx, OFFSET source
        call Str_length
        ; Returns length in RAX
        mov rsi, OFFSET source
        mov rdi, OFFSET target
        call Str_copy
        ; We just copied the string, so they should be equal.
        call Str_compare
        ; ZF = 1, strings are equal
        ; Change the first character of the target string, and
        ; compare them again.
        mov BYTE PTR [rdi], 'B'
        call Str_compare
        ; CF = 1, source < target
        mov ecx, 0
        call ExitProcess
    main ENDP
```

Actual Implementation of Procedures:

To have a complete working program, you need to provide the actual implementation of the Str_compare, Str_length, and Str_copy

procedures.

These procedures are essential for the functionality of your program. They should be implemented with appropriate assembly code to perform the desired operations.

I'll provide you with the implementation of these procedures in Irvine64 assembly:

```
; Str_compare Procedure
; Compares two strings
; Receives:
; RSI points to the source string
; RDI points to the target string
; Returns:
; Sets ZF if the strings are equal
; Sets CF if source < target

Str_compare PROC
    ; Implementation of Str_compare
    ; ...
    ret
Str_compare ENDP

; Str_length Procedure
; Gets the length of a string
; Receives: RCX points to the string
; Returns: length of string in RAX

Str_length PROC
    ; Implementation of Str_length
    ; ...
    ret
Str_length ENDP

; Str_copy Procedure
; Copies a source string to a location indicated by a target pointer
; Receives:
; RSI points to the source string
; RDI points to the target string
; Returns: nothing

Str_copy PROC
    ; Implementation of Str_copy
    ; ...
    ret
Str_copy ENDP
```

Output and Display:

In the provided test program, there is no code for displaying the results of these operations. You should add code to display whether the strings are equal, the length of the string, and the comparison results. For example, you can use WriteString and WriteDec functions to display these results:

```
mov rsi, OFFSET msg1
call WriteString  ; Display result message
call Crlf
; Check ZF and CF flags to determine equality or comparison result
; Display results accordingly
```

Irvine64 Library Setup:

The Irvine64 library needs to be included and set up properly in your assembly environment. You should have instructions at the beginning of your program to include the Irvine64 library and set it up. This usually involves specifying the paths and configurations for the Irvine64 library. Here is an example:

```
INCLUDE Irvine64.inc  ; Include Irvine64 library

.data
; Your data declarations go here

.code
main PROC
; Your program's main code goes here

main ENDP

END main
```

Actual Program:

```
INCLUDE Irvine64.inc

; -----------------------------------------------
; Str_compare
; Compares two strings
; Receives:
; RSI points to the source string
; RDI points to the target string
; Returns:
; Sets ZF if the strings are equal
; Sets CF if source < target
; -----------------------------------------------
Str_compare PROC
    USES rax rdx rsi rdi

L1:
    mov al, [rsi]
    mov dl, [rdi]
    cmp al, 0       ; End of string1?
    jne L2          ; No
    cmp dl, 0       ; Yes: End of string2?
    jne L2          ; No
    jmp L3          ; Yes, exit with ZF = 1

```

```
L2:
    inc rsi          ; Point to next
    inc rdi
    cmp al, dl       ; Characters equal?
    je L1            ; Yes, continue loop
                     ; No: Exit with flags set

L3:
    ret

Str_compare ENDP

; -----------------------------------------------
; Str_copy
; Copies a source string to a location indicated by a target pointer
; Receives:
; RSI points to the source string
; RDI points to the location where the copied string will be stored
; Returns: nothing
; -----------------------------------------------
Str_copy PROC
    USES rax rcx rsi rdi

    mov rcx, rsi    ; Get length of the source string
    call Str_length ; Returns length in RAX
    mov rcx, rax    ; Loop counter
    inc rcx          ; Add 1 for the null byte
    cld             ; Direction = up
    rep movsb        ; Copy the string
    ret
```

```
587 Str_copy ENDP
588
589 ; ---------------------------------------------
590 ; Str_length
591 ; Gets the length of a string
592 ; Receives: RCX points to the string
593 ; Returns: length of the string in RAX
594 ; ---------------------------------------------
595 Str_length PROC
596     USES rdi
597
598     mov rdi, rcx    ; Get the pointer
599     mov eax, 0      ; Character count
600 L1:
601     cmp BYTE PTR [rdi], 0  ; End of string?
602     je L2                  ; Yes: quit
603     inc rdi                ; No: Point to the next
604     inc rax                ; Add 1 to count
605     jmp L1
606 L2:
607     ret                   ; Return count in RAX
608
609 Str_length ENDP
```

```
611 .data
612     source BYTE "AABCDEFGAABCDFG",0
613     target BYTE 20 dup(0)
614
615 .code
616     main PROC
617         mov rcx, offset source
618         call Str_length      ; Returns length in RAX
619         mov rsi, offset source
620         mov rdi, offset target
621         call Str_copy
622         ; We just copied the string, so they should be equal.
623         call Str_compare
624         ; ZF = 1, strings are equal
625         ; Change the first character of the target string, and compare them again.
626         mov target, 'B'
627         call Str_compare
628         ; CF = 1, source < target
629         mov ecx, 0
630         call ExitProcess
631
632     main ENDP
633
634 END main
```

Explanation:

Irvine64 Library: The INCLUDE Irvine64.inc statement includes the Irvine64 library, providing access to Irvine's assembly functions and features.

USES Keyword: In the Str_compare and Str_copy procedures, the USES keyword is used to specify registers that will be pushed onto the stack and popped off the stack upon return from the procedure. This helps maintain the calling conventions.

Str_compare Procedure: Compares two strings pointed to by RSI and RDI. It sets the Zero Flag (ZF) if the strings are equal and the Carry Flag (CF) if the source is less than the target.

Str_copy Procedure: Copies a source string (pointed to by RSI) to a location indicated by the target pointer (RDI). It calculates the length of the source string using Str_length, then uses rep movsb to perform the copy.

Str_length Procedure: Calculates the length of a null-terminated string. It receives a pointer in RCX, and the result is returned in RAX.

.data Section: Data declarations for source and target strings.

.code Section: The main procedure demonstrates the use of these string procedures, copying the string, comparing strings, and changing a character for comparison.

This code illustrates how to use these string procedures in Irvine64 assembly, focusing on the Irvine64 register usage, stack management, and proper procedure calling conventions.

It's essential to configure your environment correctly to work with Irvine64 and ensure you have the Irvine64 library properly set up.