

Matt Skeins

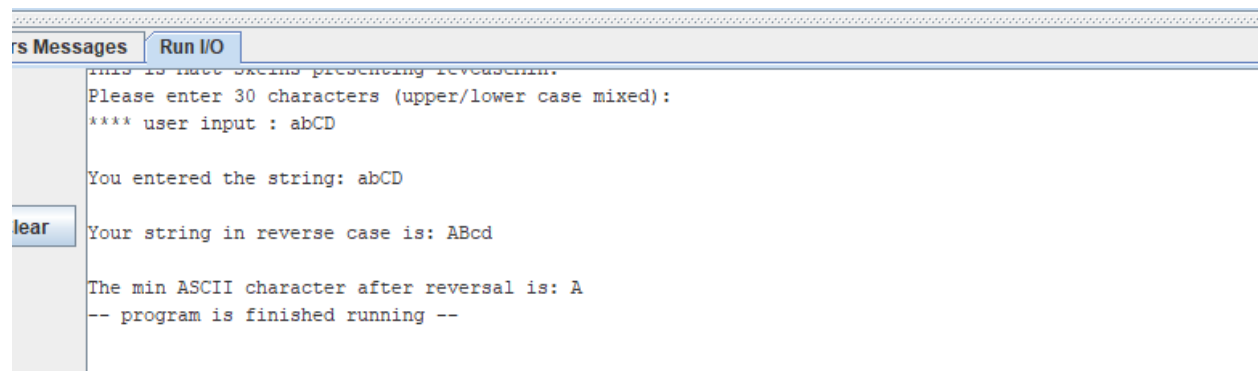
revCaseMin.asm

### Project Implementation:

To enter my revCase procedure I called `jal revCase`. This will save the return address into `$ra` (important when stack is implemented). Once inside `revCase` I needed a way to keep track of what point in each of the strings I am in. So, I initialized two “pointers” and one counter to count how many iterations have gone through. The pointers are used to show what point in the string I am in for the original string and the `revCase` string. After this occurs it goes into its first loop. The first line of this loop is the exit condition to exit when the counter reaches the end of the string (used `$a1` populated from main). Next, this loop will load the bite (character) from whatever position in the original string I am currently in. Once it is loaded I check to see if it less than 95 ascii value, which would make it capital assuming the user typed only letters. Based on if it is upper or lower case it will branch to the appropriate procedure where 32 is added or subtracted (switching cases). Once the case switch is complete I save the new character to the new string and increment my counter and pointers.

Once `reCase` has gone through the entire string it branches to the Exit procedure. Here is where I print my result from `revCase`. After that is printed I need to allocate space on the stack. I do this because I need to save the memory address of `$ra` to get back to main and also I need to return a value from `findMin`, so I allocate 2 words onto the stack. I use `jal findMin` so `$ra` has the return address of `revCase`. `findMin` operates extremely similarly to `revCase` except it only iterates through the `revCase` string. Every iteration it will check if the current value of position in the string is less than the previous value (first value is always stored before entering loop). Once the procedure is finished iterating through the string I move the lowest value to the register `$v0` to be returned. `$v0` is then saved onto the stack and I return to `revCase` with `jr $ra` since the value currently in `$ra` is `revCases` return address.

Once returned to `revCase` I load the return value back onto `$v0`, also I load the return address of main into `$ra`. Both `$ra` and `$v0` were pulled from the stack. At this point I am finished with the stack so I restore `$sp` or deallocate the stack. Now I am ready to print the value I returned from `findMin`. Once printed I return to Main using `jr $ra` since I loaded the correct memory address from the stack. At this time, we return to main where the code exits gracefully from main.

A screenshot of a terminal window with a title bar containing "rs Messages" and "Run I/O". The terminal output shows a program prompt, user input "abCD", and the program's response showing the reversed case string "ABcd" and the minimum ASCII character "A".

```
-----  
rs Messages Run I/O  
-----  
this is matt skeins presenting revcaseMin.  
Please enter 30 characters (upper/lower case mixed):  
**** user input : abCD  
  
You entered the string: abCD  
  
Your string in reverse case is: ABcd  
  
The min ASCII character after reversal is: A  
-- program is finished running --
```

## Conclusion:

I had a lot of fun working through this project. I learned the importance of correctly allocating space onto stack and deallocating the stack. This project really challenged me because I was very uncomfortable with nested procedures and working with the stack. Now that I have finished the project I can say I am pretty confident with the topic.

The biggest challenge I faced was figuring out how to return the value of findMin. For some reason I was having a hard time grasping the concept of the stack. My first few attempts of this ended with me crashing MARS several times. Once I took the time to walk through what was going on with my code and how to allocate the right way I was able to figure the error out. I had to revisit the slides several times and looked up countless examples online.