Homework 5
CS 1337

In this assignment, you will implement the following functions in ARM in the provided file.

Functions To Implement

- `char *strcat(char *dest, char *src)`
  The `strcat()` function appends the src string to the dest string, overwriting the terminating null byte at the end of dest, and then adds a terminating null byte. If dest is not large enough, program behavior is unpredictable. Returns a pointer to the resulting string dest.

- `char *strncat(char *dest, char *src, int n)`
  Same as `strcat()` except that it will use at most n bytes from src; and src does not need to be null-terminated if it contains n or more bytes. If src contains n or more bytes, `strncat()` writes n+1 bytes to dest (n from src plus the terminating null byte). Returns a pointer to the resulting string dest.

- `char *strchr(char *s, int c)`
  The `strchr()` function returns a pointer to the first occurrence of the character c in the string s. Returns a pointer to the matched character or NULL if the character is not found. The terminating null byte is considered part of the string, so that if c is specified as '\0', this function returns a pointer to the terminator.

- `char *strrchr(char *s, int c)`
  The `strrchar()` function returns a pointer to the last occurrence of the character c in the string s. Returns a pointer to the matched character or NULL if the character is not found. The terminating null byte is considered part of the string, so that if c is specified as '\0', this function returns a pointer to the terminator.

- `int strcmp(char *s1, char *s2)`
  The `strcmp()` function compares the two strings s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

- `int strncmp(char *s1, char *s2, int n)`
  Same as `strcmp()` except it compares only the first (at most) n bytes of s1 and s2.

- `char *strcpy(char *dest, char *src)`
  The `strcpy()` function copies the string pointed to by src, including the terminating null byte, to the buffer pointed to by dest. The destination string dest must be large enough to receive the copy. Returns a pointer to the destination string dest.

- `char *strncpy(char *dest, char *src, int n)`
  Same as `strcpy()` except that at most n bytes of src are copied. If there is no null byte among the first n bytes of src, the string placed in dest will not be null-terminated. If the length of src is less than n, this function will write additional null bytes to dest to ensure that a total of n bytes are written. Returns a pointer to the destination string dest.

- `char *strdup(char *s)`
  The `strdup()` function returns a pointer to a new string which is a duplicate of the string s. Memory for the new string is obtained with `malloc()`, and can be freed with `free()`. On success, returns a pointer to the duplicated string. It returns NULL if insufficient memory was available.

- `char *strndup(char *s, int n)`
  Same as `strdup()` except that at most n bytes are copied. If s is longer than n, only n bytes are copied, and a terminating null byte is added. On success, returns a pointer to the duplicated string. It returns NULL if insufficient memory was available.

- `int strlen(char *s)`
  The `strlen()` function calculates the length of the string pointed to by s, excluding the terminating null byte.

- `char *strstr(char *haystack, char *needle)`
  The `strstr()` function finds the first occurrence of the substring needle in the string haystack. The terminating null bytes are not compared. Returns a pointer to the beginning of the located substring, or NULL if the substring is not found.

Included Files

Three files are provided with the assignment: `main.c`, `my_string.h`, and `my_string.s`. The assembly source file `my_string.s` includes all 12 functions stubbed out and empty, ready to be implemented. The header file `my_string.h` includes all function declarations for the assembly source file. The C file `main.c` includes a main function to call and test all string functions. The C file will test all functions and report which tests are failing. A description of the 29 tests are below. You should not need to touch the C file, I'll be using a slightly different test file for grading.

Test Suite

1. **my_strlen(short)**
   Tests `my_strlen()` with a short string.
   Depends: `my_strlen()`.
2. **my_strlen(long)**
   Tests `my_strlen()` with a very long string.
   Depends: `my_strlen()`.
3. **my_strlen(zero)**
   Tests `my_strlen()` with a zero length string.
   Depends: `my_strlen()`.
4. **my_strcat()**
   Tests `my_strcat()`.
   Depends: `my_strcat()`, `my_strlen()`.
5. **my_strncat(n < strlen(src))**
   Tests `my_strncat()` with an n smaller than the length of the source string.
   Depends: `my_strncat()`, `my_strlen()`.
6. **my_strncat(n > strlen(src))**
   Tests `my_strncat()` with an n larger than the length of the source string.
   Depends: `my_strncat()`, `my_strlen()`.
7. **my_strchr(pos)**
   Tests `my_strchr()` with a character that exists multiple times in the search string.
   Depends: `my_strchr()`.
8. **my_strchr(neg)**
   Tests `my_strchr()` with a character that does not exist in the search string.
   Depends: `my_strchr()`.
9. **my_strchr(zero)**
   Tests `my_strchr()` with the null terminator character.
   Depends: `my_strchr()`.
10. **my_strrchr(pos)**
    Tests `my_strrchr()` with a character that exists multiple times in the search string.
    Depends: `my_strrchr()`.
11. **my_strrchr(neg)**
    Tests `my_strrchr()` with a character that does not exist in the search string.
    Depends: `my_strrchr()`.

12. **my_strrchr(zero)**
    Tests `my_strrchr()` with the null terminator character.
    Depends: `my_strrchr()`.

13. **my_strcmp(<)**
    Tests `my_strcmp()` with the first string alphabetically before the second string.
    Depends: `my_strcmp()`.

14. **my_strcmp(=)**
    Tests `my_strcmp()` with two identical strings.
    Depends: `my_strcmp()`.

15. **my_strcmp(>)**
    Tests `my_strcmp()` with the first string alphabetically after the second string.
    Depends: `my_strcmp()`.

16. **my_strncmp(<,pos)**
    Tests `my_strncmp()` with the first string alphabetically before the second string within the first n characters.
    Depends: `my_strncmp()`.

17. **my_strncmp(=,pos)**
    Tests `my_strncmp()` with the first string identical to the second string within the first n characters.
    Depends: `my_strncmp()`.

18. **my_strncmp(>,pos)**
    Tests `my_strncmp()` with the first string alphabetically after the second string within the first n characters.
    Depends: `my_strncmp()`.

19. **my_strncmp(<,neg)**
    Tests `my_strncmp()` with the first string alphabetically before the second string, but only after the first n characters.
    Depends: `my_strncmp()`.

20. **my_strncmp(>,neg)**
    Tests `my_strncmp()` with the first string alphabetically after the second string, but only after the first n characters.
    Depends: `my_strncmp()`.

21. **my_strcpy()**
    Tests `my_strcpy()`.
    Depends: `my_strcpy()`, `my_strlen()`.

22. **my_strncpy(n < strlen(src))**
    Tests `my_strncpy()` with an n smaller than the length of the source string.
    Depends: `my_strncpy()`, `my_strlen()`.

23. **my_strncpy(n > strlen(src))**
    Tests `my_strncpy()` with an n larger than the length of the source string.
    Depends: `my_strncpy()`, `my_strlen()`.

24. **my_strdup()**
    Tests `my_strdup()`.
    Depends: `my_strdup()`, `my_strlen()`, `my_strcmp()`.

25. **my_strndup(n < strlen(src))**
    Tests `my_strndup()` with an n smaller than the length of the source string.
    Depends: `my_strndup()`, `my_strlen()`, `my_strcmp()`.

26. **my_strndup(n > strlen(src))**
    Tests `my_strndup()` with an n larger than the length of the source string.
    Depends: `my_strndup()`, `my_strlen()`, `my_strcmp()`.

27. **my_strstr(pos)**
    Tests `my_strstr()` with a needle that exists within the haystack.
    Depends: `my_strstr()`, `my_strlen()`.

28. **my_strstr(neg)**
    Tests `my_strstr()` with a needle that does not exist within the haystack.
    Depends: `my_strstr()`.

29. **my_strstr(superset)**
    Tests `my_strstr()` with a needle that initially matches a substring in the haystack, but continues beyond the end of the haystack.
    Depends: `my_strstr()`.

Compile and Run

x86_64

If using Fedora on x86_64, use the following commands to compile and run the program. If using a different operating system, similar commands should exist. Seek help from classmates or the TA if experiencing difficulty.

```
$ # Compile the C and ARM files together into one binary
$ arm-none-eabi-gcc -o main main.c my_string.s --specs=rdimon.specs
$
$ # Run the compiled program to test the ARM functions
$ qemu-arm main
```

ARM

If using a Raspberry Pi (or other ARM platform), use the following commands to compile and run the program.

```
$ # Compile the C and ARM files together into one binary
$ gcc -o main main.c my_string.s
$
$ # Run the compiled program to test the ARM functions
$ ./main
```

Grading Rubric

Passing Tests: 60 points
ARM Code Quality and Efficiency: 30 points
Well Commented and Clearly Named: 10 points