Crossword check: Approx. 25 minutes

Write an x86 Assembler function that helps solving crossword puzzles: The first parameter is a string (zero-terminated) of a candidate word. The second parameter (also zero-terminated) is the "space" where this might fit in – with potentially a few letters already filled in. The function now verifies whether the first parameter matches the second:

1. They must be of exactly the same length. A length of zero, i.e. an empty string, is valid and two of these do match.
2. When the second parameter specifies a letter, this letter must appear at the same position in the first parameter.
3. If the second parameter shows a space (or whatever is defined as wildcard, see "PLACEHOLDER" below), any letter in the first parameter matches.

Write an **assembly function** that checks this and returns 0 for a match and any other value if the two words no not match.

The function's parameters and return type needs to conform to the following C function prototype:

```
int checkWord(char *candidate, char *slot);
```

You **MUST**:

- write the **implementation** of the function `checkWord` in assembly code (GNU syntax)
- conform to the "**SystemV AMD64 ABI**" **calling convention** (i.e. the one used in the course), especially for the recursive calls

You **need NOT**:

- write **other parts** of the program, like the main program, or calling this function
- perform any **error checking** on the parameters

Examples (Spaces are shown as "␣"):

- `checkWord("Student","S␣ud␣␣t")` `== 0`
- `checkWord("Student","s␣ud␣␣t")` `!= 0`
- `checkWord("Tree","S␣ud␣␣t")` `!= 0`
- `checkWord("Students","S␣ud␣␣t")` `!= 0`
- `checkWord("Student","S␣ud␣␣t␣")` `!= 0`

The following code sample shows the global constant that you should use in your implementation:

```
.equ  PLACEHOLDER,' '

      .section .text
      .type checkWord,@function
checkWord:

      # your code should start here
...
      # don't forget to return the result
```

```
.section .data
      text1: .string "Student"
      text2: .string "Students"
      text3: .string "S ud  t"
      text4: .string "s ud  t"
      text5: .string "S ud  t "

.equ  PLACEHOLDER,' '

      .section .text
      .type checkWord,@function
checkWord:
      pushq %rbp
      movq %rsp,%rbp

      # Get Parameters and initialize counter
      # Candidate     RDI
      # Slot          RSI
      movq $0,%rcx          # Current index
      movq $0,%rax          # Ensure the upper 48 Bits are empty

loop:
      movb (%rdi,%rcx,1),%al      # Get candidate letter
      movb (%rsi,%rcx,1),%ah      # Get slot letter
      incq %rcx             # Increment loop index
      cmpb $0,%al           # Candidate at end?
      je candidate_end
      cmpb $0,%ah           # Slot at end?
      je slot_end
      cmpb $PLACEHOLDER,%ah # Is it the placeholder?
      je loop
      cmpb %al,%ah          # Compare for match
      je loop
      # Difference -> No match
      # AL and AH are not zero, so we already have cor. return value
      jmp end

# Note: Both labels could be replaced by "end", as there is no
difference at all and
# the correct result is in EAX automatically

candidate_end:
      # Both are zero -> Return zero (=already in EAX)
      # AH is not zero --> Still end, return not-zero (=already in EAX)
      jmp end

slot_end:
      # Both are zero -> Return zero (=already in EAX) (cannot happen
because of previous check!)
      # AL is not zero --> Still end, return not-zero (=already in EAX)
      jmp end

end:
      movq %rbp,%rsp
      popq %rbp
      ret
```

```
.global _start
_start:
      movq $text1,%rdi
      movq $text3,%rsi
      call checkWord

      movq $0,%rdi
      cmpq $0,%rax
      je terminate
      movq $-1,%rdi
terminate:
      movq $60,%rax
      syscall
```

Recursive sum: Approx. 20 minutes

Write an x86 Assembler function which **recursively** computes the sum of natural numbers from 1 to n:

The function should return:

-1 if n < 0

0 if n = 0

n + sum(n-1) in all other cases.

The function's parameters and return type needs to conform to the following C function prototype:

```
int sum(int n);
```

You **MUST**:

- write the **implementation** of the function `checkWord` in assembly code (GNU syntax)
- conform to the "**SystemV AMD64 ABI**" **calling convention** (i.e. the one used in the course), especially for the recursive calls

You **need NOT**:

- write **other parts** of the program, like the main program, or calling this function
- perform any **error checking** on the parameters

The following code sample shows the start of the function that you should use in your implementation:

```
      .section .text
      .type sum,@function
sum:
      # your code should start here
```

```
.section .text

.type sum,@function
sum:
  pushq %rbp
  movq %rsp,%rbp
  cmpq $0,%rdi
  jl negative
  je zero
general:
  subq $8,%rsp   # Stack alignment!
  pushq %rdi
  dec %rdi
  call sum
  popq %rdi
  addq %rdi,%rax
  jmp end

negative:
  movq $-1,%rax
  jmp end

zero:
  movq $0,%rax
  jmp end

end:
  movq %rbp,%rsp
  popq %rbp
  ret


.global _start
_start:
  movq $-5,%rdi       # Test with this value: -5
#  movq $-1,%rdi      # Test with this value: -1
#  movq $0,%rdi       # Test with this value: 0
#  movq $1,%rdi       # Test with this value: 1
#  movq $5,%rdi       # Test with this value: 15
  call sum            # Call function
  movq %rax,%rdi      # Move result to return value
  movq $60,%rax       # Terminate program
  syscall
```