String Exercises

Example: Write a function to reverse the order of the words in a string. For example, if the string is **"Hello how are you"**, it will become **"you are how Hello"**.

```
void word_reverse(char *str)
{
     char sub[MAX], rev[MAX] = "";
     int i, start, end;

     // Start from the last character, and go on until the first char
     i = strlen(str);
     while (i > 0)
     {
        i--;
        // Find the end of the word (skip all blanks btw the words)
        while (str[i] == ' ')
           i--;
        end = i;
        // Find the beginning of the word
        while (i >= 0 && str[i] != ' ')
           i--;
        start = i + 1;
        // Find the word
        substr(sub, str, start, end, MAX);
        // Join it to the result
        concat(rev, rev, sub, MAX);       // strcat (rev, sub);
        // Put a blank after it
        concat(rev, rev, " ", MAX);       // strcat (rev, " ");
     }
     // Put the result into the given string
     strassign(str, rev, MAX);            // strcpy (str, rev);
}
```

- Let's write a main that inputs a string and displays its words in reverse order.

```
int main(void)
{
     char input[MAX];
     printf("Enter a string: ");
     scanline(input, MAX);
     word_reverse(input);
     printf("\nThe new string: %s\n", input);
     return(0);
}
```

Example: Write a function to find the longest common prefix in two strings, and return the result in the first string. For example, if the strings are **"How are you"**, and **"However"**, it will return **"How"** within the first string.

```
char * longest_common_prefix(char *str1, char *str2)
{
    int i = 0;
    // Find the starting point of the difference
    while ((str1[i] != '\0') && (str2[i] != '\0') &&
           (str1[i] == str2[i]))
       i++;
    // Find the longest common prefix (lcp)
    if (i > 0)
       substr(str1, str1, 0, i - 1, strlen(str1) + 1);
    // if they are completely different, lcp is null string
    else
       strcpy(str1, "");
    return (str1);
}
```

- We could also write it as a void function. The difference would be in its call. For example, the above function can be called both as a void function (returning its result in the first parameter):

```
char s1[MAX], s2[MAX];
…
longest_common_prefix(s1, s2);
printf("%s\n", s1);
```

  or as a string function:

```
printf("%s\n", longest_common_prefix(s1, s2));
```

- However, if it was a void function, it could not be called within the `printf` statement as a string function.

Pattern Matching

- One of the common operations on strings is searching for a certain string within another string. This operation is named as *Pattern Matching*.

- Let's write a function that determines the position of the first occurrence of a string within another string. For example, if the strings are **"THIS IS AN IDEA"** and **"AN"**, the function should return **8** as the starting point of **"AN"** within the string. The function should return **-1** if the first string does not exist in the second string.

```
int first_occ(char *s1, char *s2)
{
    int k, len1 = strlen(s1), len2 = strlen(s2);
    char sub[MAX];
    // Traverse s2 character by character, starting from the beginning
    for (k = 0; k <= len1 – len2; k++)
       /* Compare s2 with len2 characters of s1, starting from
          the kth position, to see if they are equal */
       if (strcmp(s2, substr(sub, s1, k, k + len2 - 1, MAX)) == 0)
          // s2 is found in s1, return the current position
          return (k);
    /* Return -1 if s2 is not found in s1 */
    return (-1);
}
```

- Notice that, if the second string is longer than the first one (i.e, if `len2` > `len1`), `len1-len2` is a negative number, so the for loop will not be entered, and the function will return **-1**.

- If we call this function as

  ```
  first_occ("THIS IS AN IDEA", "AN")
  ```

  it will return **8**. If we call it as

  ```
  first_occ("THIS IS AN IDEA", "THE")
  ```

  it will return **-1**. If we call it as

  ```
  first_occ("THIS IS AN IDEA", "IS")
  ```

  it will return **2** (not 5), because the function is not checking whether the **"IS"** found in the first string is actually the single word **"IS"**, it also finds it as a part of another word. This can easily be avoided by checking either side of the found substring, to see if they are blank. Thus, we need to modify the if statement as:

  ```
  if ((strcmp(s2, substr(sub, s1, k, k + len2 - 1, MAX)) == 0) &&
         s1[k - 1] == ' ' && s1[k + len2] == ' ')
  ```

- Notice also that, the blank checking operations will cause problems if `s2` is the first or the last word in `s1`. To include those cases, we should change our if statement as:

```
if ( (strcmp(s2, substr(sub, s1, k, k + len2 - 1, MAX)) == 0) &&
     (k == 0 || s1[k - 1] == ' ') &&
     (k == len1 – len2 || s1[k + len2] == ' ') )
```

Home Exercises:

1. Write a function that returns the first occurrence of the word `s2` within the sentence `s1`, starting at a certain position `pos`.
2. Write a function that returns the last occurrence of a word within a sentence

Example: Write a function that returns all of the occurrences of a word within a sentence. For example, if the sentence is **"She is very good very very nice"**, and the word is **"very"**, it will return 7, 17 and 22.

- Since it is not possibble to return more than one results with the `return` statement, we need to use an array as an output parameter. We should also return the number of occurrences of the word as an output parameter.

```
void all_occ(char *s1, char *s2, int occ[], int *cnt)
{
    int k = 0,
        len1 = strlen(s1),
        len2 = strlen(s2);
    char sub[MAX];

    *cnt = 0;
    while (k <= len1 - len2)
        if ( (strcmp(s2, substr(sub, s1, k, k + len2 - 1, MAX)) == 0) &&
             (k == 0 || s1[k - 1] == ' ') &&
             (k == len1 - len2 || s1[k + len2] == ' ') )
        {
           occ[*cnt] = k;
           *cnt += 1;
           k += len2 + 1; // jump to the next word
        }
        else
           k++; // move to the next character
}
```

Example: Write a function to delete the first occurrence of a word in a sentence. For example, if the sentence is **"THIS IS AN IDEA"**, the word is **"AN"**, the sentence will become **"THIS IS IDEA"**.

```
void del_first_occur(char *str, char *myword)
{
    char before[MAX],
         after[MAX];
    int  len1 = strlen(str),
         len2 = strlen(myword);

    // Find the position of the first occurrence
    int pos = first_occ(str, myword);

    // If the word occurs in the string
    if (pos != -1)
       // Join the parts before and after the word
        concat(str, substr(before, str, 0, pos - 1, MAX),
                    substr(after, str, pos + len2 + 1, len1 - 1, MAX), MAX);
}
```

Home Exercise:

3. Write a function that deletes the last occurrence of a word from a sentence.

Example: Write a function to replace the first occurrence of a word with another word in a sentence. For example, if the sentence is **"THIS IS AN IDEA"**, the first word is **"AN"**, the second word is **"THE"**, the sentence will become **"THIS IS THE IDEA"**.

```
void replace_first_occur(char *str, char *myword, char *repwith)
{
    char before[MAX], after[MAX], temp[MAX];
    int len1 = strlen(str),
        len2 = strlen(myword);

    // Find the position of the first occurrence
    int pos = first_occ(str, myword);
    // If the word occurs in the string
    if (pos != -1)
    {
       // Join the part before the word and the new word
        concat(temp, substr(before, str, 0, pos - 1, MAX),
                     repwith, MAX);
       // Join the part after the word
        concat(str, temp, substr(after, str, pos + len2, len1 - 1, MAX),
                    MAX);
    }
}
```

Home Exercise:

4. Write a function that replaces the last occurrence of a word with another word within a sentence.

Example: Write a function to delete all occurrences of a word in a sentence. For example, if the sentence is **"THIS IS A VERY VERY GOOD IDEA"**, the word is **"VERY"**, the sentence will become **"THIS IS A GOOD IDEA"**.

```
void del_all_occur(char *str, char *myword)
{
    char sub[MAX], temp[MAX];
    int pos[MAX], cnt, k;
    int len1 = strlen(str),
        len2 = strlen(myword);

    // Find the positions of all occurrences
    all_occ(str, myword, pos, &cnt);

    // If the word occurs in the string
    if (cnt != 0)
    {
       // Put the part before the first occurrence
       substr(temp, str, 0, pos[0] - 1, MAX);
       for (k = 1; k < cnt; k++)
          // Join the part between two occurrences
          concat(temp, temp,
                 substr(sub, str, pos[k - 1] + len2 + 1, pos[k] - 1, MAX),
                 MAX);

       // Join the part after last occurrence
      concat(str, temp,
             substr(sub, str, pos[cnt - 1] + len2 + 1, len1 - 1, MAX),
             MAX);
    }
}
```

Home Exercise:

5. Write a function that replaces all occurrences of a word with another word within a sentence.