String Manipulating Functions

- C language provides built-in functions in its its string handling library. This library, with the header file **<string.h>**, provides many useful functions for manipulating string data.

- `strlen` function takes a string as a parameter and returns the number of characters in the string not including NULL.

Example:

```
char str[20]= "Example";
printf("%d", strlen(str);
```

will output 7.

Example: Write a function to replace a character with another character in a string.

```
void replace(char *str,                // The given string
             char c1, char c2) {       // The characters
     int i;
     for (i = 0; i < strlen(str); i++)
         if (str[i] == c1)
             str[i] = c2;
}
```

- What is the problem with the loop segment of the `replace` function?

- Let's write a main that inputs a sentence and two characters, and replaces all occurences of the first character with the second one within the sentence.

```
int main(void) {
     char sent[20], ch1, ch2;
     // Get the sentence and the characters
     printf("Enter a sentence: ");
     scanf("%[^\n]", sent);      // gets(sent);
     printf("Enter the character to replace: ");
     scanf(" %c", &ch1);
     printf("Replace with: ");
     scanf(" %c", &ch2);
     // Make replacements and display the new sentence
     replace(sent, ch1, ch2);
     printf("\nThe new sentence: %s\n", sent);
     return (0);
}
```

Output:

```
Enter a sentence: Good Morning
Enter the character to replace: o
Replace with: *

The new sentence: G**d M*rning
```

Home Exercise: Write a function that takes a string as input parameter and returns the number of uppercase letters, lowercase letters, digits, and whitespace characters in it.

- We can assign a string to a character array during its declaration as in the above example. But, we are not allowed to make such an assignment later in the program. Thus,

```
char str[20];
str = "Example";
```

  is a compilation error, because as you know, an array name with no subscript represents the address of the initial array element.

- We need to use `strcpy` function in such a case:

```
char *strcpy(char *sl, const char *s2);
```

  copies the string `s2` into the array `s1`. The value of `s1` is returned.

```
char *strncpy(char *sl, const char *s2, size_t n);
```

  copies at most `n` characters of the string `s2` into the array `s1`. The value of `s1` is returned.

- The first argument of `strcpy` must be large enough to store the string and the terminating NULL character of the second argument.
- The keyword `const` in the second parameter means that the function can not change the contents of that string.
- The `strncpy` function does not copy the terminating NULL character if the third argument is less than the length of the string in the second argument. So, we need to put it at the end of the first argument with an assignment statement.

Example:

```
char y[25], z[15], x[] = "Happy Birthday to you";
printf("x = %s\ny = %s\n", x, strcpy(y, x));
strncpy(z, x, 14);
z[14] = '\0';
printf("z = %s\n", z);
printf("y = %s\n", strncpy(y, z, 20));
```

Output:

Example: Write a function to reverse a string using pointers. For example, if the string is **"Hello how are you"**, it will become **"uoy era woh olleH"**.

```
void reverse(char *str) {
     char rev[MAX];
     int i, k = 0;
     int len = strlen(str);
     for (i = len-1; i >= 0; i--) {
          *(rev+k) = *(str+i);
          k++;
     }
     *(rev+k) = '\0';
     strcpy(str, rev); // Copy the result into the given string
}
```

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

```
int main(void) {
     char sent[MAX];
     printf("Enter a string: ");
     gets(sent);
     reverse(sent);
     printf("\nReverse is: %s\n", sent);
     return(0);
}
```

- Joining two strings operation is named as *concatenation* operation. This is a very frequently used operation when dealing with strings.

```
char *strcat(char *sl, const char *s2);
```

appends the string `s2` to the array `s1`. The first character of `s2` overwrites the terminating NULL character of `s1`. The value of `s1` is returned.

```
char *strncat(char *s1, const char *s2, size_t n);
```

appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating NULL character of `s1`. The value of `s1` is returned.

Example:

```
char sl[20] = "Very Happy ";
char s2[] = "New Year";
char s3[40] = "";
printf("%s\n", strcat(sl, s2));
printf("%s\n", strncat(s3, sl, 5));
printf("%s\n", strcat(s3, sl));
```

Output:

- Remember that the characters can be compared using relational operators, and the result is decided according to their ASCII values.

- The C language does not allow us to use relational operators to compare strings. It provides two functions for this purpose:

  ```
  int strcmp(const char *sl, const char *s2)
  ```

  compares the string `s1` with the string `s2`. The function returns **0** if `s1` is equal to `s2`, **-1** if `s1` is less than `s2`, **1** if `s1` is greater than `s2`.

  ```
  int strncmp(const char *sl, const char *s2, size_t n)
  ```

  compares the first `n` characters of the strings `s1` and `s2`. The function returns **0**, **-1** or **1** depending on the result of the comparison, as `strcmp`.

- The results of the comparisons are determined by comparing the two strings character by character.

Example:

```
char s1[20] = "Happy New Year";
char s2[20] = "Happy New Year";
char s3[20] = "Happy Holidays";
printf("%3d%3d%3d\n", strcmp(s1, s2), strcmp(s1, s3), strcmp(s3, s2));
printf("%3d%3d%3d\n", strncmp(s1, s3, 5), strncmp(s3, s1, 7),
                      strncmp(s2, s3, 9));
```

Output:

➢ *READ Sec. 8.6 and 8.7 from Deitel & Deitel.*