Tutorial on Sorting

January 9, 2024

1 Sorting Strings

In this tutorial we will write some code to sort arrays on strings. Before starting: how do we decide if one string is smaller than another?

1.1 Standard Library String Comparison Function `strcmp`

The standard library function `strcmp(s, t)` compares strings `s` and `t`, and returns:

- Negative if `s` is lexicographically less than `t`
- Zero if `s` is lexicographically equal to `t`
- Positive if `s` is lexicographically greater than `t`

What does "lexicographically" mean? Examples:

1. "abc" is greater than "abb"
2. "C" is less than "Cpp"
3. ...

but see `https://en.wikipedia.org/wiki/Lexicographic\_order` for details.

1.2 Writing a String Sorting Algorithm

Your first task is to write a function `sort` that sorts an array of strings.

- Use the same algorithms as in the lab on sorting, but use `strcmp` instead of `>`
- For example, `strcmp(s, t) > 0` means `s` is greater than `t` (in dictionary order).

See the file `tutorial_sort_template.c` for some initial code.

2 Writing Your Own String Comparison Function

2.1 Working with Strings in C

Initializing a string as a character array:

```
char some_string[] = "something something something"
```

The variable `some_string` points to an array with just enough space to hold the sequence of characters 's', 'o', ..., and a final character '\0', which indicates the end of the string.

Instead of passing the length of a string, we look for '\0' to know where a string ends. For instance:

```
void my_print(char* s){
    int i = 0;
    while(s[i] != '\0'){
        printf("%c", s[i]);
        i++;
    }
}

// ...USING POINTERS
void my_print2(char* s){
 while(*s != '\0'){
  printf("%c", *s);
  s++;
 }
}
```

Since `some_string` is an array, we can change its values like any other array:

```
some_string[0] = 'S';
```

Instead, if you create a character pointer and set its value:

```
char* my_string = "this is my string!";
```

then `my_string` points to a string constant `"this is my string"`, which you can't modify. **Note**: in C++ we should write

```
const char* my_string = "this is my string!";
```

as omitting const will trigger an error. Note that we can not modify a character pointer created in this way, but we can reassign it:

```
my_string[0] = "s"; // ERROR
my_string = "another string"; // OK
```

Here is another example of working with strings (we propose three different ways of doing the same thing):

```
// my_strcpy: copy t to s
void my_strcpy(char *s, char *t){
    int i = 0;
    while(t[i] != '\0'){
        s[i] = t[i];
        i++;
    }
}

// EQUIVALENT FOR LOOP
void my_strcpy2(char *s, char *t){
    for(int i = 0; t[i] != '\0'; i++){
        s[i] = t[i];
    }
}

// ASSIGN AND COMPARE AT THE SAME TIME
void my_strcpy3(char *s, char *t){
    int i = 0;
    while((s[i] = t[i]) != '\0'){
        i++;
    }
}
```

2.2 Writing `my_strcmp`

Recall that 'a' returns the ASCII code for the character 'a'. The value of 'b' is 'a' + 1, the value of 'c' is 'a' + 2, and so on. The value returned by `strcmp` is the difference of the ASCII codes of the first different characters.

Examples:

- `strcmp("abc", "abb")` returns 'c' - 'b', which is 1
- `strcmp("C", "Cpp")` returns '\0' - 'p', which is -'p'.

Complete the definition of the `my_strcmp` function provided in `tutorial_sort_template.c` and test it.