Tutorial 7: sorting

Brendan Murphy

January 25, 2023

Contents

1 Sorting strings

How do we decide if one string is less than another?

1.1 Lexicographical order

`https://en.wikipedia.org/wiki/Lexicographic_order`

Examples:

1. "abc" is greater than "abb"
2. "C" is less than "Cpp"
3. think of more examples...

1.2 Standard library string comparision 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* than `t`
- *positive* if `s` is lexicographically *greater* than `t`

1.3 Writing a string sorting algorithm

Write a function `sort` that sorts an array of strings.

- use the same algorithms as the lab, but use `strcmp` instead of `>`
- e.g. `strcmp(s,t) > 0` means `s` is greater than `t` (in dictionary order).

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';
```

If you create a character pointer and later set its value:

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

then `my_string` points to a *string constant* `"this is my string"`, which you can't modify.

Here is another example of working with strings (there are three versions that do the same thing in different ways):

```
// 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++;
    }
}
```

Note: in Python, you can use the "walrus" operator to assign and compare at the same time:

```
if (val := some_function()) > 0:
    print("The value ", val, " is greater than 0")
```

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 first different characters.

Examples:

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

```
int my_strcmp(char* s, char* t){
    // YOUR CODE HERE
}
```