Strings (3)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

A good question from last week

- We used `const` for `char *s`
- Why can we still do `s++` inside the function?
- Ans: the `const` here doesn't prevent `count_spaces` from modifying `s`; it prevent the function from modifying what `s` points to.

```
int count_spaces(const char *s){
     int count = 0, i;

     for (; *s != '\0'; s++)

        if (*s == ' ')
          count++;
     return count;
}
```

Quick Recap of last week

- Reading and Writing Strings
- Accessing the Characters in a String
- Using the C String Library
  - `strcpy`
  - `strlen`

Please take some minutes to recall these topics in your brain

slido

Identify which calls don't write a single new-line character? (multiple choice)

ⓘ Start presenting to display the poll results on this slide.

Outline

- Using the C String Library
  - String Concatenation
  - String Compare
- Arrays of Strings

The `strcat` (String Concatenation) Function

- "`apple`" and "`pen`" after string concatenation will become "`apple pen`"
- We can use `strcat` to do string concatenation in C
- `strcat` appends the contents of the string `s2` to the end of the string `s1`.

```
char *strcat(char *s1, const char *s2); //function prototype

strcpy(str1, "abc");
strcat(str1, "def"); /* str1 now contains "abcdef" */
```

The `strcat` (String Concatenation) Function

- `strcat` returns `s1` (a pointer to the resulting string), but the value returned by strcat is normally discarded (will not use another string variable to store the return value)
- The following example shows how the return value might be used
  - use `strcat` to concatenate **multiple** strings

```
strcpy(str1, "abc");
strcpy(str2, "def");
strcat(str1, strcat(str2, "ghi"));
/* str1 now contains "abcdefghi"; str2 contains "defghi" */
```

The `strcat` (String Concatenation) Function

- To solve this problem, we can use `strncat`, which is a safer but slower version of `strcat`.
- Like `strncpy`, it has a third argument that limits the number of characters it will copy.
- However, `strncat` will terminate `str1` with a null character, which isn't included in the third argument
  - there are still some possibilities that the null character will exceed the length of `str1`
  - How to setup the third argument to avoid this problem?

```
strncat(str1, str2, ___?___ );
```

The `strcat` (String Concatenation) Function

- `sizeof(str1) - strlen(str1)`: the amount of space remaining in `str1`
  - `sizeof(str1)`: the number of bytes (characters) in the char array
    - the size of one character is one byte, so the returned value here is equal to the lenght of the char array
  - `strlen(str1)`: the length of string (number of character) stored in the char array
- `- 1`: subtracts 1 for the null character

```
strncat(str1, str2, sizeof(str1) - strlen(str1) - 1);
```

The `strcmp` (String Comparison) Function

- **Next**, we will introduce function that help us compare strings
- `strcmp` compares the strings `s1` and `s2`, returning a value less than, equal to, or greater than 0, depending on whether `s1` is less than, equal to, or greater than `s2`.
  - if `s1` is less than `s2`: return value < 0
  - if `s1` is equal to `s2`: return value = = 0
  - if `s1` is greater than `s2`: return value > 0

```
int strcmp(const char *s1, const char *s2);
```

The `strcmp` (String Comparison) Function

- By choosing the proper operator (<, <=, >, >=, ==, !=), we can test any possible relationship between `str1` and `str2`.
- However, what is the meaning of "`str1` is less than `str2`"?

```
if (strcmp(str1, str2) < 0)     /* is str1 < str2? */
if (strcmp(str1, str2) <= 0) /* is str1 <= str2? */
```

The `strcmp` (String Comparison) Function

- `strcmp` considers `s1` to be **less** than `s2` if either one of the following conditions is satisfied
  - The first *i* characters of `s1` and `s2` match, but the (*i*+1)st character of `s1` is less than the (*i*+1)st character of `s2`.
    - For example, "`abc`" is less than "`bcd`" and "`abe`"
  - All characters of `s1` match `s2`, but `s1` is shorter than `s2`.
    - For example, "`abc`" is less than "`abcd`"

The `strcmp` (String Comparison) Function

- As it compares two strings, `strcmp` looks at the **numerical** codes for the characters (e.g., ASCII code) in the strings.
- Some knowledge of the underlying character set is helpful to predict what `strcmp` will do.
- Important properties of ASCII:
  - A–Z, a–z, and 0–9 have consecutive codes.
  - All upper-case letters are less than all lower-case letters.
  - Digits are less than letters.
  - …

Arrays of Strings

- In prior examples, we only use the string variable containing a **single** string
- What if I want to store **mulitple** strings?
  - For example, I want an array storing the name of all classmates.
- How to store an array of strings?
- There is more than one way to store an array of strings.

"Hello World"

Arrays of Strings

- One option is to use a two-dimensional array of characters, with one string per row:
- The number of rows in the array can be omitted, but we must specify the number of columns.
  - **number of rows**: the number of strings in this array, which is **obvious** by looking at the initializer
  - number of columns: the **length** of **string** for each element, which we need to specified first

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```

Arrays of Strings

- Unfortunately…
- the `planets` array contains a fair bit of wasted space (extra null characters)
- because **not all strings in the array have equal length.**
- Most collections of strings will have a mixture of long strings and short strings.
- Therefore, we often use another ways to create an array of strings.

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

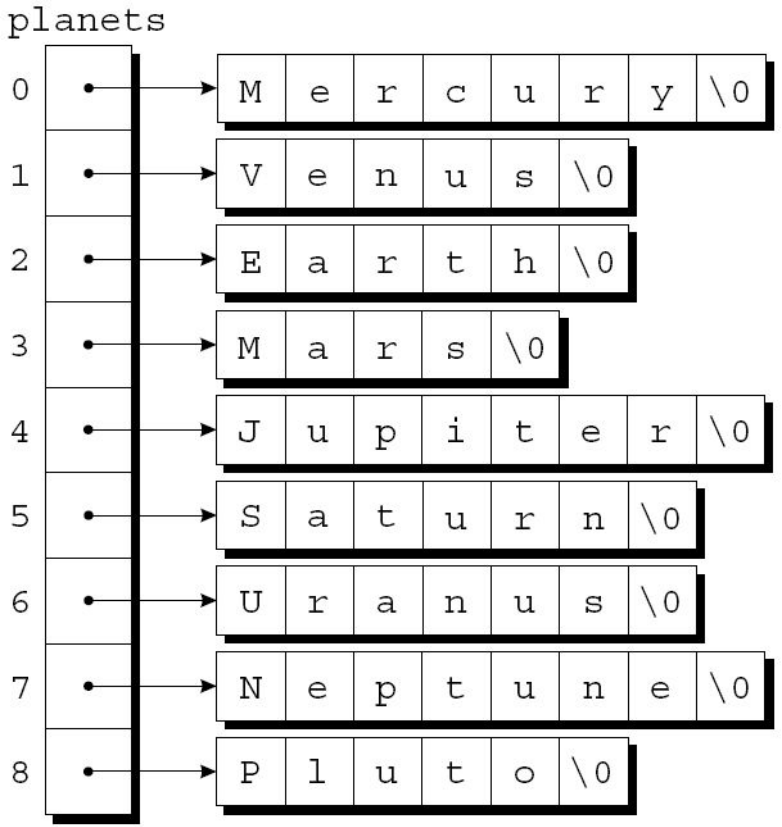
Arrays of Strings

- What we need is a ***ragged array,*** whose rows can have different lengths.
  - ragged (adj.): having an irregular or uneven surface, edge, or outline.
- We can simulate a ragged array in C by creating an array whose elements are *pointers* to strings:

```
// char planets[][8] = …
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
```

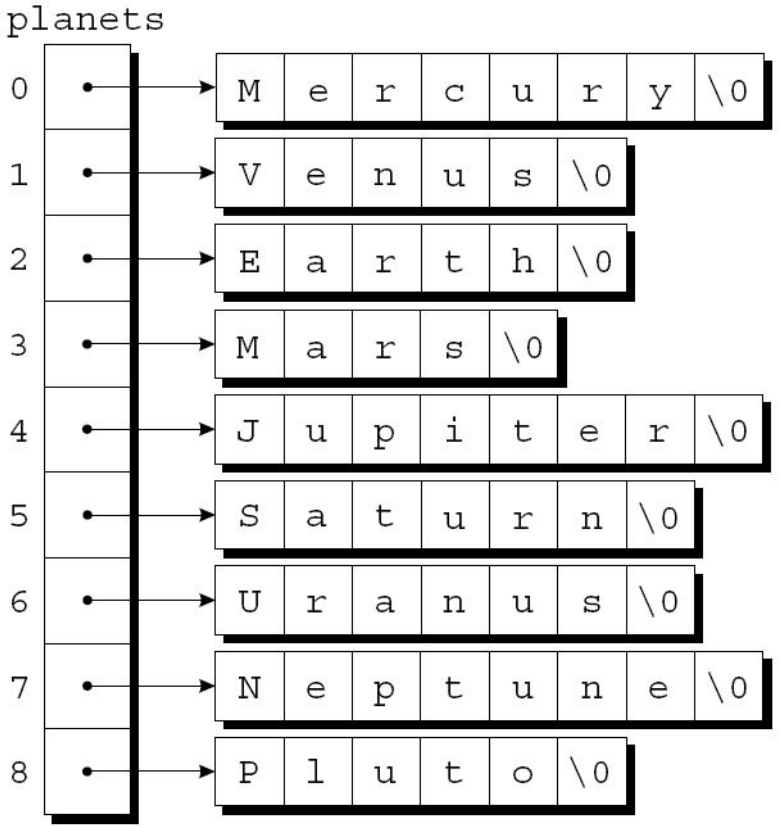
Arrays of Strings

- This small change has a dramatic effect on how `planets` is stored
- Each element is a pointer to a null-terminated string
- No longer any wasted characters in the strings
  - Although we still need to allocate spcace for the pointers
  - Still less wasted space compared to 2D array version in most conditions



Arrays

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0



Arrays of Strings

- To access one of the planet names, all we need do is **subscript** the `planets` array.
- Accessing a character in a planet name is done in the same way as accessing an element of a two-dimensional array.

```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
printf("%s", planets[0]);
```

Mercury

Arrays of Strings

- Moreover, we can also access the character of each string stored in the array
- For example, a loop that searches the planets array for strings beginning with the letter `M`
- Why `sizeof(planets)/sizeof(planets[0])` instead of `sizeof(planets)`?

```
for (int i = 0; i <  sizeof(planets)/sizeof(planets[0]); i++)
        if (planets[i][0] == 'M')
            printf("%s begins with M\n", planets[i]);
```

```
Mercury begins with M
Mars begins with M
```

Summary

- Using the C String Library
  - String Concatenation: `strcat` and `strncat`
  - String Compare: `strcmp` and `strcmp`
- Arrays of Strings
  - `char planets[][NUM_STRING]`
  - `char *planets[]`