

Strings in C

CPSC 275
Introduction to Computer Systems

Introduction

- Strings are arrays of characters in which a special character—the **null** character—marks the end.
- Strings may be defined as *string literals* (*string constants*) or *string variables*.
- The C library provides a collection of functions for working with strings.

2

String Literals

- A **string literal** is a sequence of characters enclosed within double quotes.
- String literals may contain escape sequences.
- Examples:
"Sue Smith"
"123"
`printf("sum = %d\n", sum);`

3

How String Literals Are Stored

- When a C compiler encounters a string literal of length n in a program, it sets aside $n + 1$ bytes of memory for the string.
- This memory will contain the characters in the string, plus one extra character—the **null character**—to mark the end of the string.
- The null character is a byte whose bits are all zero, so it's represented by the `\0` escape sequence.

4

How String Literals Are Stored

- The string literal "abc" is stored as an array of four characters:

a	b	c	\0
---	---	---	----

- The string "" is stored as a single null character:

\0

5

How String Literals Are Stored

- Since a string literal is stored as an array, the compiler treats it as a pointer of type `char *`.
- So, what is the type of the first argument of `printf`?
- We can use a string literal wherever C allows a `char *` pointer:
`char *p;`
`p = "abc";`

6

String Literals versus Character Constants

- A string literal containing a single character isn't the same as a character constant.
 - "a" is represented by a *pointer*.
 - 'a' is represented by an *integer*.
- A legal call of `printf`:
`printf("\n");`
- An illegal call:
`printf('\n');` **** WRONG ****

7

String Variables

- If a string variable needs to hold 80 characters, it must be declared to have length 81:
`#define STR_LEN 80`
...
`char str[STR_LEN+1];` */* Why? */*
- Be sure to leave room for the null character when declaring a string variable.

8

```
int mystery(char a[])
{
    int i;
    for (i = 0; a[i] != '\0'; i++)
        /* do nothing */;
    return i;
}
```

- What does this function return?
- The actual length of a string depends on the position of the terminating null character.
- Will it always work?

9

Initializing a String Variable

- A string variable can be initialized at the same time it's declared:
`char date1[8] = "June 14";`
- The compiler will automatically add a null character so that `date1` can be used as a string:

date1

J	u	n	e		1	4	\0
---	---	---	---	--	---	---	----

10

Initializing a String Variable

- If the initializer is too short to fill the string variable, the compiler adds extra null characters:

```
char date2[9] = "June 14";
```

Appearance of `date2`:

date2

J	u	n	e		1	4	\0	\0
---	---	---	---	--	---	---	----	----

11

Initializing a String Variable

- An initializer for a string variable can't be longer than the variable, but it can be the same length:
`char date3[7] = "June 14";`
- There's no room for the null character, so the compiler makes no attempt to store one:

date3

J	u	n	e		1	4
---	---	---	---	--	---	---

12

Initializing a String Variable

- The declaration of a string variable may omit its length, in which case the compiler computes it:
`char date4[] = "June 14";`
- The compiler sets aside eight characters for `date4`, enough to store the characters in "June 14" plus a null character.

13

Character Arrays vs Character Pointers

- The declaration
`char date[] = "June 14";`
declares `date` to be an *array*,
- The similar-looking
`char *date = "June 14";`
declares `date` to be a *pointer*.

14

Character Arrays vs Character Pointers

- However, there are significant differences between the two versions of `date`.
 - In the array version, the characters stored in `date` can be modified.
 - In the pointer version, `date` points to a string literal that shouldn't be modified.

15

Character Arrays vs Character Pointers

- The declaration
`char *p;`
does not allocate space for a string.
- Before we can use `p` as a string, it must point to an array of characters.
- One possibility is to make `p` point to a string variable:
`char str[STR_LEN+1], *p;
p = str;`
- Another possibility is to make `p` point to a dynamically allocated string. (How?)

16

What's wrong with this code?

```
char *p;  
  
p[0] = 'a';    /** WRONG **/  
p[1] = 'b';    /** WRONG **/  
p[2] = 'c';    /** WRONG **/  
p[3] = '\0';   /** WRONG **/
```

17

Writing Strings

- The `%s` conversion specification allows `printf` to write a string:
`char str[] = "Are we having fun yet?";
printf("%s\n", str);`
The output will be
Are we having fun yet?
- `printf` writes the characters in a string one by one until it encounters a null character.

18

Reading Strings

- The `%s` conversion specification allows `scanf` to read a string into a character array:
`scanf("%s", str);`
- `str` is treated as a pointer, so there's no need to put the `&` operator in front of `str`. (why not?)
- When `scanf` is called, it skips white space, then reads characters and stores them in `str` until it encounters a white-space character.
- `scanf` always stores a null character at the end of the string.

19

Reading Strings Using `scanf`

- Consider the following program fragment:

```
char sentence[SENT_LEN+1];  
printf("Enter a sentence:\n");  
scanf("%s", sentence);
```
- Suppose that after the prompt
Enter a sentence:
the user enters the line
To C, or not to C: that is the question.
- `scanf` will store the string "To" in `sentence`.

20

```
void mystery2(char str[], int n)  
{  
    int ch, i = 0;  
    while ((ch = getchar()) != '\n')  
        if (i < n)  
            str[i++] = ch;  
    str[i] = '\0';  
}
```

What does this function do?
It reads a line of characters.

Why is the statement
`str[i] = '\0';`
necessary?

21

Accessing the Characters in a String

```
int mystery3(char s[])  
{  
    int k = 0, i;  
    for (i = 0; s[i] != '\0'; i++)  
        if (s[i] == ' ')  
            k++;  
    return k;  
}
```

What does this function do?
It returns the number of white spaces in a string.

22

Accessing the Characters in a String

- A version that uses pointer arithmetic instead of array subscripting :

```
int count_spaces(char *s)  
{  
    int count = 0;  
    for (; *s != '\0'; s++)  
        if (*s == ' ')  
            count++;  
    return count;  
}
```

23

Accessing the Characters in a String

- Is it better to use array operations or pointer operations to access the characters in a string? We can use either or both. Traditionally, C programmers lean toward using pointer operations.
- Should a string parameter be declared as an array or as a pointer? There's no difference between the two.

24



Exercise 1: Computing the Length of a String

Write a C function:

```
int mystrlen(char *s);
```

which will return the length of `s`, that is, the number of characters between the beginning of the string and the terminating null character.

Exercise 2: Copying Strings

Write a C function:

```
char *mystrcpy(char *dest, char *src);
```

which will copy characters from `src` to `dest` strings and return `dest`. Make sure `dest` is null-terminated.

Exercise 3: Concatenating Strings

Write a C function:

```
char *mystrcat(char *dest, char *src);
```

which will append characters from `src` to the end of `dest` and return `dest`. Make sure `dest` is null-terminated.