

# CSE105 – Structured Programming

## **String**

# Fundamentals of Characters and Strings

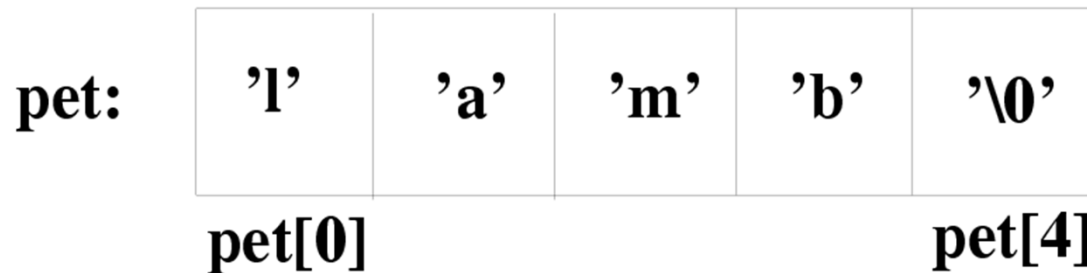
- **Character constants:** (which actually represents an **integer value**)  
'a', 'A', '0', '1', '\n', ' ', '\0', ..
- **String constants:**  
"I am a student", "Your score %d\n"
- **Character variable to store a character**  
`char c = 'l';`
- **Character array, which can store a string**  
`char str[15] = "I am a student";`

# String Implementation and Representation

- Implemented by an array of characters

```
char pet[5] = {'l', 'a', 'm', 'b', '\0'};  
printf("Mary has a little %s.\n", pet);
```

- More accurate: NULL-terminated array of characters



- Represented by a character array

```
char color[] = "blue";
```

# What is String

- Strings are arrays of characters in which a special character—the null character ('\0')—marks the end.
- The C library provides a collection of functions for working with strings.

# String Literals

- A ***string literal*** is a sequence of characters enclosed within double quotes:

"When you come to a fork in the road, take it."

- String literals may contain escape sequences.
- Character escapes often appear in `printf` and `scanf` format strings.

- For example, each `\n` character in the string

"Candy\nIs dandy\nBut liquor\nIs quicker.\n --Ogden Nash\n"

causes the cursor to advance to the next line:

```
Candy
Is dandy
But liquor
Is quicker.
  --Ogden Nash
```

# Continuing a String Literal

- The backslash character (\) can be used to continue a string literal from one line to the next:

```
printf("When you come to a fork in the road, take it. \n\n--Yogi Berra");
```

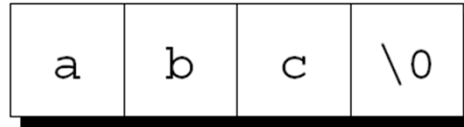
- In general, the \ character can be used to join two or more lines of a program into a single line.

# 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.

# How String Literals Are Stored

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



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





# 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 ** */
```

# String Variables

- Any one-dimensional array of characters can be used to store a string.
- A string must be terminated by a null character.

# String Variables

- If a string variable needs to hold 80 characters, it must be declared to have length 81:

```
char str[80+1];
```

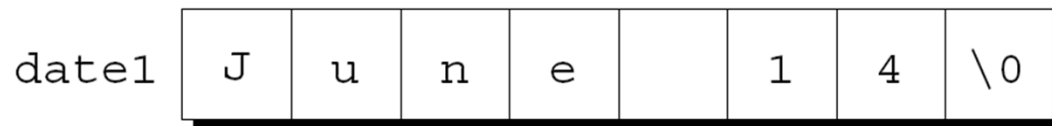
- Adding 1 to the desired length allows room for the null character at the end of the string.
- Failing to do so may cause unpredictable results when the program is executed.

# 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:



# 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:

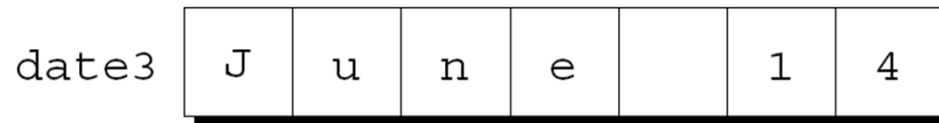
date2	J	u	n	e		1	4	\0	\0
-------	---	---	---	---	--	---	---	----	----

# 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:



# 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.
- Omitting the length of a string variable is especially useful if the initializer is long, since computing the length by hand is error-prone.

# Reading and Writing Strings

- Writing a string is easy using either `printf` or `puts`.
- To read a string in a single step, we can use either `scanf` or `gets`.
- As an alternative, we can read strings one character at a time.



# Writing Strings Using `printf` and `puts`

- 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.

# Writing Strings Using `printf` and `puts`

- `printf` isn't the only function that can write strings.
- The C library also provides `puts`:  
`puts(str);`
- After writing a string, `puts` always writes an additional new-line character.

# Reading Strings Using `scanf` and `gets`

- 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`.
- 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.

# Reading Strings Using `scanf` and `gets`

- `scanf` won't usually read a full line of input.
- A new-line character will cause `scanf` to stop reading, but so will a space or tab character.
- To read an entire line of input, we can use `gets`.
- Properties of `gets`:
  - Doesn't skip white space before starting to read input.
  - Reads until it finds a new-line character.
  - Discards the new-line character instead of storing it; the null character takes its place.

# Reading Strings Using `scanf` and `gets`

- Consider the following program fragment:

```
char sentence[80+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`.

# Reading Strings Using `scanf` and `gets`

- Suppose that we replace `scanf` by `gets`:  
`gets(sentence) ;`
- When the user enters the same input as before, `gets` will store the string  
" To C, or not to C: that is the question."  
in `sentence`.

# Reading Strings Character by Character

- Suppose we need a function that (1) doesn't skip white-space characters, (2) stops reading at the first new-line character (which isn't stored in the string), and (3) discards extra characters.
- A prototype for the function:  

```
int read_line(char str[], int n);
```
- If the input line contains more than `n` characters, `read_line` will discard the additional characters.
- `read_line` will return the number of characters it stores in `str`.

# Reading Strings Character by Character

- `read_line` consists primarily of a loop that calls `getchar` to read a character and then stores the character in `str`, provided that there's room left:

```
int read_line(char str[], int n)
{
    int i = 0;
    char ch;
    while ((ch = getchar()) != '\n')
        if (i < n)
            str[i++] = ch;
    str[i] = '\0';    /* terminates string */
    return i;         /* number of characters stored */
}
```



# Reading Strings Character by Character

- Before returning, `read_line` puts a null character at the end of the string.
- Standard functions such as `scanf` and `gets` automatically put a null character at the end of an input string.
- If we're writing our own input function, we must take on that responsibility.

# Accessing the Characters in a String

- Since strings are stored as arrays, we can use subscripting to access the characters in a string.
- To process every character in a string `s`, we can set up a loop that increments a counter `i` and selects characters via the expression `s[i]`.

# Accessing the Characters in a String

- A function that counts the number of spaces in a string:

```
int count_spaces()  
{  
    char s[1000];  
    int count = 0, i;  
    gets(s);  
    for (i = 0; s[i] != '\0'; i++)  
        if (s[i] == ' ')  
            count++;  
    return count;  
}
```