Handling Character Strings

String Literals

- A string literal is enclosed within double quotes.
- Eg., format string in calls to `printf/scanf`.
- String literal may contain escape sequences as characters.

```
#include <stdio.h>
int main() {
    char a[] = "As you sow\nSo, shall\nYou reap\n";
    printf("%s", a);
}
```

Handling Character Strings

String Literals

- Hexadecimal and octal escape sequence are also valid.
- Octal sequence ends with 3 digit or first non-octal digit, eg.,
  - "\1234" ≡ "\123" and "4" and
  - "\458" ≡ "\45" and "8".
- Hexadecimal sequence not limited by 3 digits, ends on encountering first non-hexadcimal digit.

Handling Character Strings

String Literals

- Hexadecimal escape sequences should be used with caution.
  - Eg., \x22 represents ",
  - But \x22{a-f} could represent valid other characters.
  - So, how a string of the form `"Can ...` can be represented?
- Partitioning the string literal and writing adjacent to one another is equivalent to concatenation.
- Eg. `char a[] = "\x22" "Can you program in C?\x22";` produces "Can you program in C?".

Handling Character Strings

- If literal is too long to fit into a line, "\" is used to partition the string, but second part must begin from first column of next line, e.g.,
```
printf("A quick brown fox jumped over \
the lazy dog.");
```
- "\" is used to join two or more lines of code a standard C process of splicing.
- It messes up the program indentation.
- So, it is better to use adjacent literal separated by only one white space, and let compiler to join these strings.
- Eg.
```
printf("A quick brown fox jumped over"
       "the lazy dog.")
```

Handling Character Strings

Operation on Literals

- A literals is stored as a null terminated character array, e.g.:

H	e	l	l	o	\0	x	x	x

- Can be used wherever `char *` is allowed.
- Eg., a literal can appear in RHS of an assignment: `char *p; p = "abc";`
- Subscripting on literals is also permitted: `"abc"[1]` represents `b`

Handling Character Strings

Operation on Literals

A function for converting 0-15 into hexdecimal digit:

```
#include <stdio.h>
char convertHex(int d) {
    return "0123456789abcdef"[d];
}
int main() {
   int n;
   printf("Enter a number 0-15: ");
   scanf("%d", &n);
   printf("%c\n", convertHex(n));
}
```

- `scanf` treats white space as the end of the string.
- Null character `\0` is automatically inserted in a string constant.
- But, it should be explicitly inserted into a user created string.

Handling Character Strings

String Variables

- C uses 1D null terminated character arrays for string variables.
- But such character arrays can also be used in conventional way.
- E.g.: `char date[9] = "April 17"` will be stored as:

A	p	r	i	l		1	7	\0

- C views string literals as initilizers.

Handling Character Strings

String Variables

- One could use the following initializer to same effect:
  `char date[9] = {'A','p','r','i','l',' ','1','7', '\0'}`
- But the use of string initializers is simple, it automatically pads null characters when smaller initializers are provided.
- Eg. `char date[9] = "May 17"` will be stored as:

M	a	y		1	7	\0	\0	\0

- If an initializer is longer, null character will be dropped, and character array becomes unusable as a string.

Handling Character Strings

Character Arrays vs. Character Pointers

- In a character array, stored elements can be modified.
- But the string being pointed to by a pointer is a literal, so it can not be modified.
- Whereas a pointer can be made to point to other literals during execution.
- So, if a string is to be modified, an array should be set aside to store the same.
- Declaration `char *p` causes compiler to only set memory for storing address, not space for storing string.

Reading and Writing String

Printing Strings

- Possible format conversions for `printf` are `"%s"` and `"%.ns"`:
  - Prints characters until hitting "\0" in first case.
  - With format `"%.ns"`, prints n characters if string length is $> n$, the full string if length $< n$.
- C string library also provides `puts` for printing strings:
  - It just takes one argument `puts(str)`
  - After printing `str`, `\n` is printed.