CSI 333 – Lecture 8
Introduction to C: Part VII (More on Strings)
Ref: Chapter 8 of Deitel & Deitel.

String: Sequence of characters ending with ’\0’.

(a) Array version:

```c
char greet[] = "Hello";
```

```
0 1 2 3 4 5
H e l l o \0 greet
```

After the assignments

```c
```

```
0 1 2 3 4 5
H e l p s \0 greet
```
(b) Pointer Version:

```c
char *gptr = "Hello";
```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>

**Pointer Arithmetic:** Recall that

```c
char a[10];
```

assigns 10 consecutive bytes to `a[0]`, `a[1]`, …, `a[9]`. Now,

Type of `a[0]` : char  
Type of `&a[0]` : char *

So, the following assignment is valid:

```c
char *px;  px = &a[0];
```
Effect:

The assignment

\[ px = &a[0]; \]

is equivalent to

\[ px = a; \]

Reason: In C, the name of an array is a synonym for the address of the zeroth location (i.e., starting address) of the array.
Important Note

Although the name of an array is an address, it is a constant; it cannot be changed. Hence, statements such as

\[
a++; \quad a = px;
\]

are not allowed.

Correspondence Summary:

- \&a[0] : Synonymous with a
- a[0] : Synonymous with *a
- \&a[i] : Synonymous with a+i
- a[i] : Synonymous with *(a+i)
Because of the correspondence, the statement

\[ a[8] = '!' \]

is equivalent to

\[ *(a+8) = '!' \]

Also, the statement

\[ px = a + 8; \]

is equivalent to

\[ px = a; px += 8; \]
Strings and Pointer Arithmetic (continued)

Sample program segment:

```c
char a[10]; char *px = a;
*(a+8) = ’!’; px += 8;
printf("%c %c\n", a[8], *px);
```

Output:

! !

Summary: After the code segment

```c
char a[10]; char *px = a;
```

the characters in `a` can be accessed as

- `a[0], a[1], …, a[9]`
- `*a, *(a+1), …, *(a+9)`
- `*px, *(px+1), …, *(px+9)`
Original Example: (from Slide 8-3)

```c
char  *gptr = "Hello";
*(gptr+3) = 'p';  *(gptr+4) = 's';
```

Program examples: (for pointer arithmetic)

Example 1: Computing the length of a string. (The length does not include the ’\0’ char.)
(a) Array version:

```c
size_t strlen (const char *s) {
    int i = 0;
    while (s[i] != ’\0’) 
        i++;
    return (size_t) i;
} /* End of strlen. */
```

Notes:

- The type `size_t` is usually defined to be `unsigned int` or `unsigned long`.
- The `const` qualifier indicates that string `s` is not modified by the function.
- As a function parameter, “`char *s`” and “`char s[]`” are equivalent.
- A typecast is used in the `return` statement.
(b) **Pointer version:**

```c
size_t strlen (const char *s) {
    char *p = s;
    while (*p != '\0')
        p++;
    return (size_t) (p-s);
} /* End of strlen */
```

**Notes:**

- The statement “char *p = s;” initializes the pointer p.
- The loop could also have been written as follows:
  ```c
  while (*p)
      p++;
  ```
  **Reason:** The character ‘\0’ has the (ASCII) value zero. So, the loop would end as soon as that character is seen.
- The expression (p−s) gives the length of the string.
Example 2: Copying a string:

```c
void copy_str (char d[], char s[])
```

(a) **Array version:** Exercise.

(b) **Pointer version:**

```c
void copy_str (char d[], char s[]) {
    while ( *(d++) = *(s++) )
    ; /* Empty loop body. */
} /* End of copy_str */
```

**Notes:**

- The function uses the fact that an assignment statement is an expression. (The assignment statement ensures that the ‘\0’ character is also copied.)

- Loop terminates just after copying ‘\0’, since the (ASCII) value of that character is zero.
Functions in `<string.h>`

**Ref:** Sections 8.6, 8.7 and 8.8 of Deitel & Deitel. (Better descriptions in Harbison & Steele’s book.)

**Important Note**

Functions in `<string.h>` expect strings terminated with ‘`\0`’.

**Some Examples:**

**Function:**

```c
char *strcpy (char *s1, const char *s2)
```

**Description:** Copies the string `s2` to `s1`. Returns a pointer to `s1`. (The ‘`\0`’ character is also copied.)
Function: char *strncpy (char *s1, const char *s2, size_t n)

Description:

- Copies at most n chars from s2 to s1. Returns a pointer to s1.
- If s2 has n or more chars, then the first n chars are copied to s1.
- If s2 has less than n chars, then all the chars of s2 are copied to s1. The remaining spots are filled with ‘\0’.
Example:

Initial:

\[
\begin{array}{c|c|c|c|c|c|c|c}
 & x & y & z & w & q & \backslash 0 \\
\hline
s1 &  &  &  &  &  & \\
\hline
 & a & b & c & \backslash 0 \\
\hline
s2 &  &  &  &  &  & \\
\end{array}
\]

After `strncpy(s1, s2, 3)`:

\[
\begin{array}{c|c|c|c|c|c|c|c}
 & a & b & c & w & q & \backslash 0 \\
\hline
s1 &  &  &  &  &  & \\
\hline
 & a & b & c & \backslash 0 \\
\hline
s2 &  &  &  &  &  & \\
\end{array}
\]

After `strncpy(s1, s2, 5)`:

\[
\begin{array}{c|c|c|c|c|c|c|c}
 & a & b & c & \backslash 0 & \backslash 0 & \backslash 0 \\
\hline
s1 &  &  &  &  &  & \\
\hline
 & a & b & c & \backslash 0 \\
\hline
s2 &  &  &  &  &  & \\
\end{array}
\]
Function:

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

- Compares strings `s1` and `s2`.
- Returns zero if strings are identical.
- Returns a negative (< 0) value if `s1` precedes `s2` in dictionary order.
- Returns a positive (> 0) value if `s1` follows `s2` in dictionary order.

Recall

The operation `(s1 == s2)` does not compare the strings.
**Function** `strtok`: Useful in breaking up a string into tokens.

**Example:** The following string has 4 tokens:

"Here is a string"

**Typical usage of `strtok`:**

```c
char s[] = "Here is a string"; char *tptr;

/* First call to `strtok`. */
tptr = strtok (s, " ");

/* Subsequent calls. */
while (tptr != NULL) {
    printf("%s\n", tptr); tptr = strtok (NULL, " ");
}
```
Notes regarding `strtok`:

- The first call is different from the subsequent calls.
- The function modifies the argument string.

String conversion functions:

- Header: `<stdlib.h>`
- Convert a string to a number.

Some Examples:

- Function: `int atoi (const char *nptr)`
- Returns the integer value of the (digit) string specified by `nptr`.
Functions in <string.h> (continued)

Usage:

```c
int i;
i = atoi("-2375");  printf("%d\n", i);
```

Output:

```
-2375
```

Similarly,

```c
atof : Converts from string to double.
atol : Converts from string to long.
```
Search Functions for Strings

Ref: Section 8.8 of Deitel & Deitel. (Better descriptions in Harbison & Steele’s book.)

Header: `<string.h>`

Example 1:

- **Function:** `char *strchr (const char *str, int c)`
- Tries to locate the **first** occurrence of the char specified by `c` in the string `str`.
- If found, returns a pointer to that char; otherwise, returns `NULL`. 
Usage of strchr:

```c
char *str = "Mean:251.7";
char *p; p = strchr (str, ':');
```

Example 2:

- Function: `char *strrchr (const char *str, int c)`
- Similar to strchr, except that it locates the last occurrence.
Example 3:

- Function:
  ```c
  char  *strstr (const char *s1, const char *s2)
  ```
- Tries to locate the **first** occurrence of string `s2` in string `s1`.
- If found, returns a pointer to the string in `s1`; otherwise, returns `NULL`. 
Usage of `strstr`:

```c
char *s1 = "An Example"; char *s2 = "amp";
char *p; p = strstr (s1, s2);
```

Reading Assignment:
- Review Handout 7.1 which uses the `strstr` function.
- Study the other string functions explained in Chapter 8 of Deitel & Deitel.