C, string.h

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 9 due Monday on Gradescope
- Lab 11 next Tuesday (can check off for full credit by 12/5)

$ man strsep.

Also maybe one char or an array of char

```
char *strsep(char **s, const char *d);
```

→ a pointer to one character, or an array of pointers

It's changable!

→ It could be:

ptr → ptr → char ⇒ a waste of space.

ptr → array of chars ✓

array → pointers to char

array of arrays of chars

} maybe not an array of sth. because there is no length provided. How many pointers do I have?

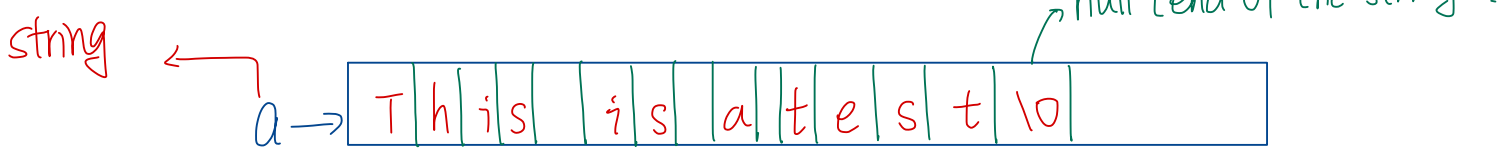
✅ When a function returns a pointer, it usually means TWO things:

① The function allocated memory for you (via malloc).
It created new memory on the heap.
Now you own it → you must free it later.
Example idea: makeList() returns a newly allocated list.

② The function is returning something you already own.
You gave it memory (a buffer, a struct, etc.).
It used or modified the memory.
It returns a pointer into that same memory.
You still free it only once, when you're done.

❌ What it NEVER means:
It NEVER returns a pointer to a local variable on the stack.
Stack memory disappears after the function ends → pointer becomes invalid.

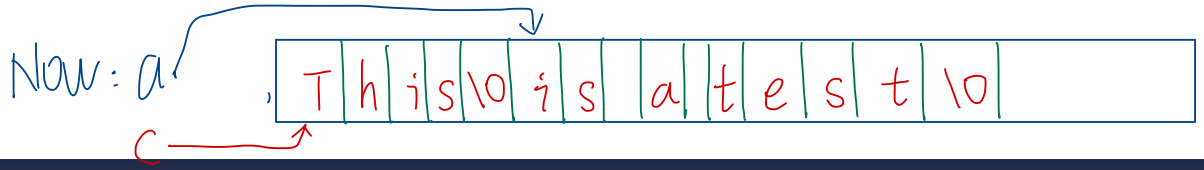
char *strsep(char **s, const char *d);



I should tell it where the variable "a" lives so that they can change it.

c = strsep(&a, b)

run through the string "a" and look for anything in "b" so it will find the first space in "a" (which is what "b" has) and then replace it with "\0"



char *strsep(char **s, const char *d);

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = "This is a test";
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    return 0;
}
```

segmentation fault! Why?

"char *s" is a string → read only memory.

so we need to put this string on heap.

char *strsep(char **s, const char *d);

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a test");
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    return 0;
}
```

malloc some space for me, put the string there, and gave me a ponter to that string.

Everything good now?

No. I forget to free it.

char *strsep(char **s, const char *d);

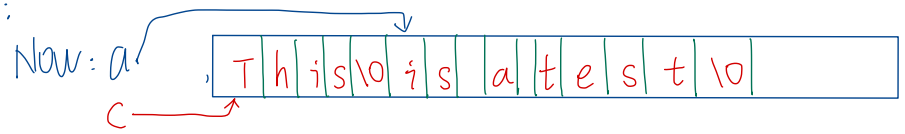
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a test");
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    free(s);
    return 0;
}
```

How about this version?

```
This
free(): invalid pointer
Aborted (core dumped)
```

recall:



What we malloc at very begining (a) now points to other place. No longer the same pointer we got from malloc() So I can not free it because malloc didn't give me that pointer.

char *strsep(char **s, const char *d);

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a test");
    char *deleteme = s;
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    puts(s);
    free(deleteme);
    return 0;
}
```

We use a new pointer to remember the address of original "a". Now it works.

char *strsep(char **s, const char *d);

DESCRIPTION

If *stringp is NULL, the **strsep**() function returns NULL and does nothing else. Otherwise, this function finds the first token in the string *stringp, that is delimited by one of the bytes in the string delim. This token is terminated by overwriting the delimiter with a null byte ('\0'), and *stringp is updated to point past the token. In case no delimiter was found, the token is taken to be the entire string *stringp, and *stringp is made NULL.

RETURN VALUE

The **strsep**() function returns a pointer to the token, that is, it returns the original value of *stringp.

what happens if no delimiter is found?

In that situation, the token is considered to be the entire remaining string, and the String P (the pointer to the rest of the string) is set to NULL.

This means that once we reach the end of the input and no further delimiter exists, the entire string is returned as the final token, and the pointer to the remaining substring becomes NULL because there is nothing left to process.

So effectively, the tokenizer says:
“Here is the last token, and there's no remaining string.”

This behavior allows us to perform some useful operations with token parsing.

char *strsep(char **s, const char *d);

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a test");
    char *deleteme = s;
    char *d = " ";
    while (s) {
        char *token = strsep(&s, d);
        puts(token);
    }

    free(deleteme);
    return 0;
}
```

It will give you:

```
This
is
a
test
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a  test");
    char *deleteme = s;
    char *d = " ";
    while (s) {
        char *token = strsep(&s, d);
        puts(token);
    }

    free(deleteme);
    return 0;
}
```

It will give you:

```
This
is
a

test
```

man string.h

```
DESCRIPTION
       Some  of  the  functionality described on this reference page extends the ISO C standard. Applications shall define
       the appropriate feature test macro (see the System Interfaces volume of POSIX.1-2017, Section 2.2, The  Compilation
       Environment) to enable the visibility of these symbols in this header.

       The <string.h> header shall define NULL and size_t as described in <stddef.h>.

       The <string.h> header shall define the locale_t type as described in <locale.h>.

       The  following  shall be declared as functions and may also be defined as macros. Function prototypes shall be pro-
       vided for use with ISO C standard compilers.
```

① ISO: International Standards Orgnization. They standarlized the C language.

② There might be things that are not in the standard, but meet the standard, and possibly extend it.

③. NULL: A pointer to memory position zero.

④. size_t: the size of size_t depends on the system. It always the same width as a pointer.

⑤. locale_t: the user's locale, meaning regional settings.