The Common String Library Functions

`strtok(buffer, delimiters)`

A “token” in `buffer` is defined to be a sequence of characters between any two occurrences of characters in delimiters. A call to `strtok()` places a null character at the end of the first “token” and returns the address of the first character of the “token”. Subsequent calls to `strtok()` with a NULL as the first parameter will find and isolate each “token” in `buffer`.

Parameters	`buffer`	a null-terminated string
	`delimiters`	a null-terminated string. The characters in the string mark the beginning and end of “tokens” in `buffer`.
Return value	The address of the next “token” in `buffer`	

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

#define MAX 80

int main(void)
{
    char *Token = NULL;
    char Buffer[MAX] = {};

    printf("Enter 3 words separated by commas ");
    fgets(Buffer, MAX-1, stdin);

    Token = strtok(Buffer, ",");
    printf("The first word is %s\n", Token);
    Token = strtok(NULL, ",");
    printf("The second word is %s\n", Token);
    Token = strtok(NULL, ",");
    printf("The third word is %s\n", Token);

    return 0;
}
```

`strtok()` returns a `char` pointer so we need a variable to hold that address

```c
char *Token = NULL;
```

`strtok()` takes 2 parameters

First parameter is a string

We declared the array `Buffer` and used `fgets()` to make a string (null terminated array)

Second parameter is the string of delimiters

this can a quoted string

or

it can be a null terminated array (string)

After the `fgets()`

```
(gdb) p Buffer
$1 = "a,b,c\n", '\000' <repeats 73 times>

(gdb) p &Buffer
$3 = (char (*)[80]) 0x7fffffffe6d0
```

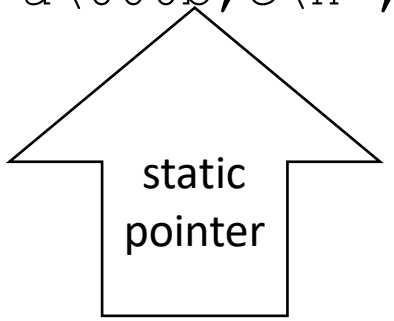
```
Token = strtok(Buffer, ",");
```

`strtok()` will find the first occurrence of the delimiter in `Buffer`. It will replace the delimiter with `NULL`.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```

`strtok()` then creates an internal static pointer that points to the address of the character just past the newly inserted `NULL`. We cannot see/access this pointer.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```



This static pointer is separate from the `NULL` the delimiter was replaced with previously.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>

(gdb) p &Buffer
$1 = (char (*)[80]) 0x7fffffffe6d0
```

`strtok()` returns the pointer to the token

```
(gdb) p Token
$4 = 0x7fffffffe6d0 "a"
```

For the first call to `strtok()`, this address is same as the address of the string being tokenized.

```
Token = strtok(Buffer, ",");
```

We can this print the token

```
printf("The first word is %s\n", Token);
```

Remember that `printf()` with `%s` is looking for a pointer/address of a null terminated string.

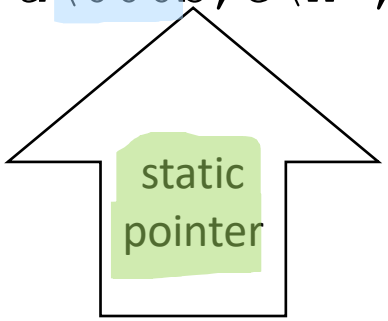
The second time we call `strtok()` to get the next token, we call it with a first parameter of `NULL` instead of the string (like we did the first time).

```
Token = strtok(NULL, ",");
```

When you pass it `NULL` on the second call, you are signaling to it to use that internal static pointer as its starting point for looking for the next delimiter.

This usage of `NULL` is not referring to the `NULL` that `strtok()` put in place of your delimiter in the string itself.

```
(gdb) p Buffer
$5 = "a\000b,c\n", '\000' <repeats 73 times>
```

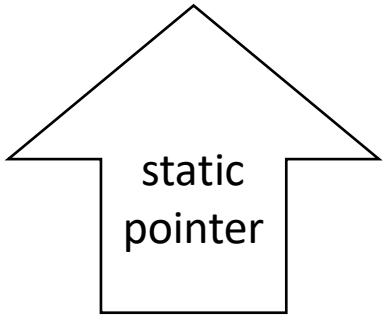


The second time we call `strtok()` to get the next token, we call it with a first parameter of `NULL` instead of the string (like we did the first time).

```
Token = strtok(NULL, ",");
```

This replaces the second delimiter with a `NULL` and moves the static pointer to the next character after the newly inserted `NULL`.

```
(gdb) p Buffer
$2 = "a\000b\000c\n", '\000' <repeats 73 times>
```



The address of the second token is returned by `strtok()` and stored in `Token`.

If you pass the string in any call to `strtok()` after the first one, then you are signaling to `strtok()` that you are starting over with a new string and that it should not use the static pointer it had from the previous call.

```
Token = strtok(Buffer, ",");
printf("The first word is %s\n", Token);
Token = strtok(Buffer, ",");
printf("The second word is %s\n", Token);
Token = strtok(Buffer, ",");
printf("The third word is %s\n", Token);

[frenchdm@omega ~]$ a.out
Enter 3 words separated by commas a,b,c
The first word is a
The second word is a
The third word is a
[frenchdm@omega ~]$
```

strtok()

```
printf("\nEnter a line of text (max of %d) using Delimiters %s\n\n",
       MAX_INPUT-1, Delimiters);
fgets(Buffer, MAX_INPUT, stdin);
Buffer[strlen(Buffer) - 1] = '\0';

Token = strtok(Buffer, Delimiters);

while (Token != NULL)
{
   printf("Token = %s\n", Token);
   Token = strtok(NULL, Delimiters);
}
```

strtok()

```
{Austin|817-DOG-1234|10}
{Jenny|867-5309|40}
{Prof French|817-272-0161|162}
{Fake Name|123-456-7890|-1}
```

strtok()

Enter the phrase like {Name|Phone|Age} to be tokenized **{Austin|817-DOG-1234|10}**
Hello Austin - your phone number is 817-DOG-1234 and you are 10 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Jenny|867-5309|40}**
Hello Jenny - your phone number is 867-5309 and you are 40 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Prof French|817-272-0161|162}**
Hello Prof French - your phone number is 817-272-0161 and you are 162 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Fake Name|123-456-7890|-1}**
Hello Fake Name - your phone number is 123-456-7890 and you are -1 years old

```
char *Delimiters = "{}|";
char *Token = NULL;
char TokenPhrase[PHRASELEN] = {};
char Name[PHRASELEN] = {};
char Phone[PHRASELEN] = {};
int age = 0;

printf("Enter the phrase like {Name|Phone|Age} to be tokenized ");
fgets(TokenPhrase, PHRASELEN-1, stdin);
TokenPhrase[strlen(TokenPhrase)-1] = '\0';

Token = strtok(TokenPhrase, Delimiters);
strcpy(Name, Token);

Token = strtok(NULL, Delimiters);
strcpy(Phone, Token);

Token = strtok(NULL, Delimiters);
age = atoi(Token);

printf("Hello %s - your phone number is %s and you are %d years old\n",
        Name, Phone, age);
```

strtok()

Enter the phrase like {Name|Phone|Age} to be tokenized **{Austin|817-DOG-1234|10}**
Hello Austin - your phone number is 817-DOG-1234 and you are 10 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Jenny|867-5309|40}**
Hello Jenny - your phone number is 867-5309 and you are 40 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Prof French|817-272-0161|162}**
Hello Prof French - your phone number is 817-272-0161 and you are 162 years old

Enter the phrase like {Name|Phone|Age} to be tokenized **{Fake Name|123-456-7890|-1}**
Hello Fake Name - your phone number is 123-456-7890 and you are -1 years old