計算機程式設計

Computer Programming

Strings 2

Instructor: 林英嘉

2024/12/02

W13 Slido: #3000360

GitHub repo

Outline

- Pointers (mainly related to strings with recaps)
- Common functions for strings
  - Buffer overflow

[Recap] Definition of Pointers

- A pointer is a **variable** that is used to store the memory address of another variable.
- A pointer also has a data type, which is the type of the pointed variable.
- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1  int main(void){
2      int *iptr; // Declaration of a pointer
3      int i = 10;
4      iptr = &i;
5  }
```

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1  int main(void){
2      int *iptr; // Declaration of a pointer
3      int i = 10;
4      iptr = &i;
5  }
```

Line 4:

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1  int main(void){
2      int *iptr; // Declaration of a pointer
3      int i = 10;
4      iptr = &i;
5  }
```

Line 4:

(1) Use ampersand (&) to get the address of i

[Declaration] Make a pointer point to a variable

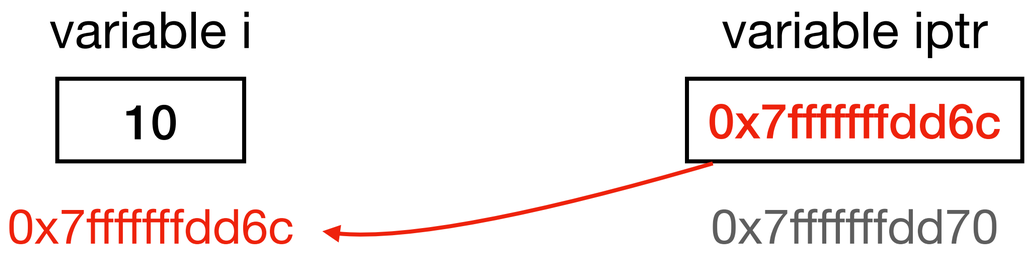
To build the relationship between a pointer and a variable, we can perform the following operations:

```
1  int main(void){
2      int *iptr; // Declaration of a pointer
3      int i = 10;
4      iptr = &i;
5  }
```

Line 4:

(1) Use ampersand (&) to get the address of i

(2) Assigning the address of **i** to the pointer



[Recap, Declaration] Pointer to an Array

C-course-materials/06-Pointers-arrays/declaration_and_print.c

We can create a pointer to an array with any initial address of an pointer:

```
int arr[5] = {1, 2, 3, 4, 5}, *p;
p = &arr[0];
```

equal to `p = arr;`

Or you can create a pointer for an array in one line:

```
int arr[5] = {1, 2, 3, 4, 5};
int *p = &arr[0];
```

equal to `int *p = arr;`

Character Pointers

[Declaration] Pointers for strings

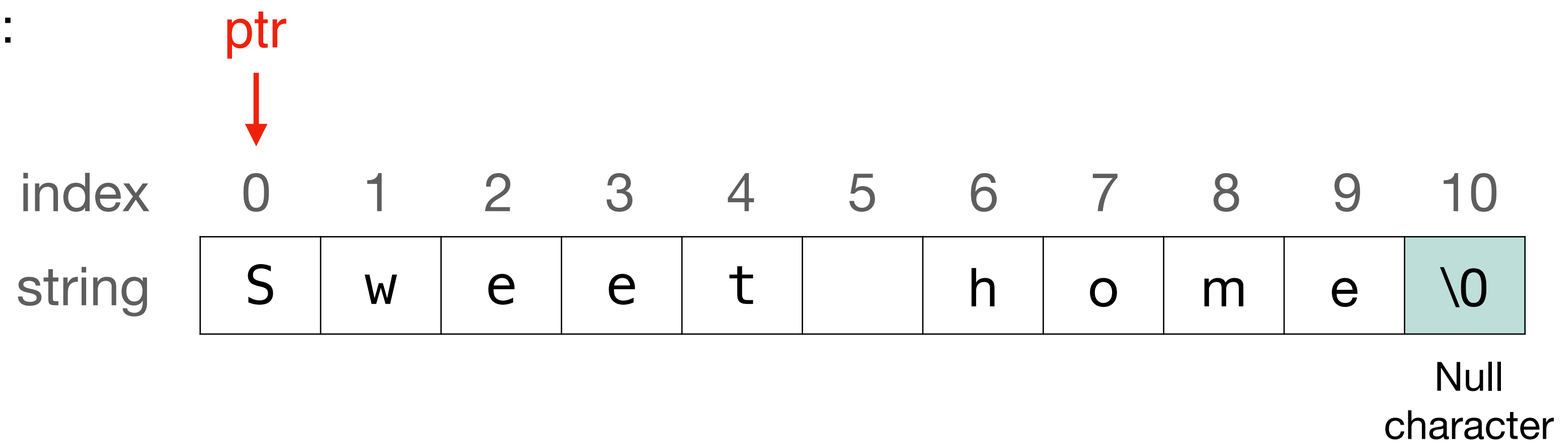
1. Pointer declaration with **a character array**

```
char a_string[] = "Sweet home";
char *ptr1 = a_string;
```

2. Pointer declaration with **a string literal (字串常量)**

```
char *ptr2 = "Sweet home";
```

Illustration for both 1 and 2:



Print an Array via a Pointer

C-course-materials/06-Pointers-arrays/print_str_via_ptr.c

Using Pointer Arithmetic (both 1 and 2👌)

```c
#include <stdio.h>
int main(void){
    char a_string[] = "Sweet home";
    char *ptr = a_string;
    for (int i = 0; i < sizeof(a_string); i++){
        printf("%c", *(ptr + i));
    }
}
```

```c
#include <stdio.h>
int main(void){
    char a_string[] = "Sweet home";
    char *ptr = a_string;
    while(*ptr != '\0'){
        printf("%c", *ptr);
        ptr++;
    }
}
```

Modify the content of a string via a pointer

C-course-materials/06-Pointers-arrays/modify_str.c

We **cannot modify** the string from 2. Pointer declaration with **a string literal (字串常量)**, which is static.

```c
#include <stdio.h>
int main(void){
    char a_string[] = "Sweet home";
    char *ptr1 = a_string;
    char *ptr2 = "Sweet home";
    *(ptr1+6) = 'd';
    *(ptr2+6) = 'd'; // This line causes an error!
    for (int i = 0; i < 10; i++){
        printf("%c", ptr1[i]);
    }
}
```

Output

```
Sweet dome
```

[Summary] Declaration of Pointers for strings

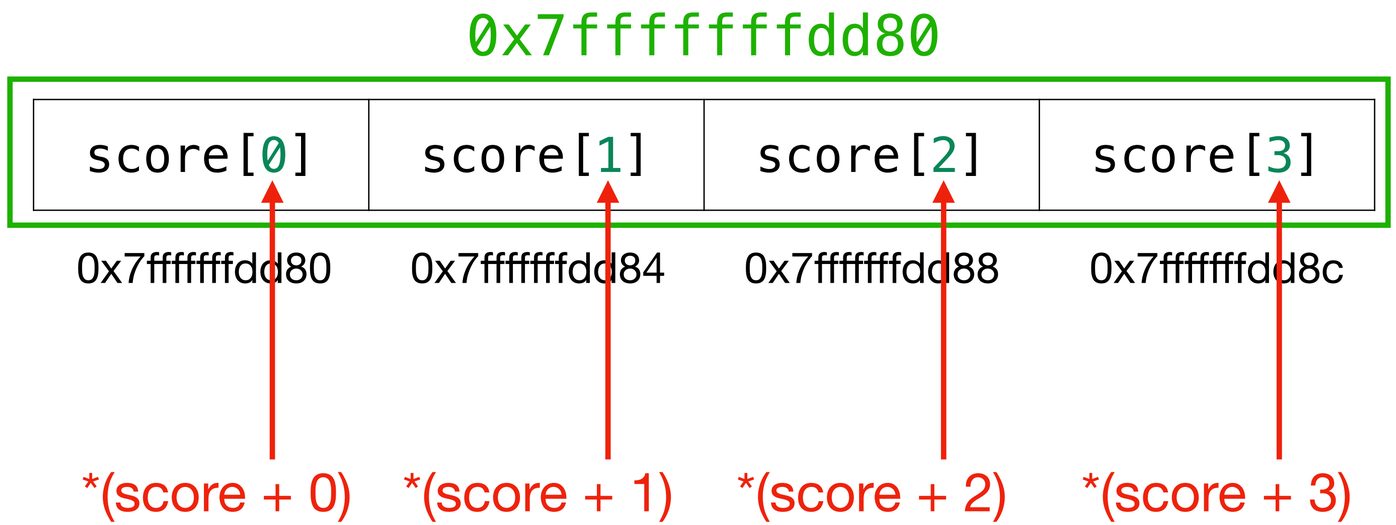
Declaration	Example	String Modifiable?
1. Pointer declaration with a character array	`char a_string[] = "Sweet home";` `char *ptr1 = a_string;`	Yes
2. Pointer declaration with a string literal	`char *ptr2 = "Sweet home";`	No (static)

Array / Pointer Subscripting

[Usage] Array / Pointer Subscripting

```
int arr[5] = {1, 2, 3, 4};
int *p = &arr[0];
```

Assume we have an array or a pointer variable called A:
subscripting A[B] is equal to the operation of *(A+B).



[Usage] Pointer Subscripting

C-course-materials/06-Pointers-arrays/subscripting.c

```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4};
    int *p = &arr[0];
    printf("arr[0] = %d\n", arr[0]);
    printf("p[0] = %d\n", p[0]);
    printf("arr[1] = %d\n", arr[1]);
    printf("p[1] = %d\n", p[1]);
}
```

Output

```
arr[0] = 1
p[0] = 1
arr[1] = 2
p[1] = 2
```

Print an Array via a Pointer

C-course-materials/06-Pointers-arrays/print_str_via_ptr.c
C-course-materials/06-Pointers-arrays/str_ptr_subscripting.c

Using Pointer Arithmetic

```c
#include <stdio.h>
int main(void){
    char a_string[] = "Sweet home";
    char *ptr = a_string;
    for (int i = 0; i < sizeof(a_string); i++){
        printf("%c", *(ptr + i));
    }
}
```

```c
#include <stdio.h>
int main(void){
    char a_string[] = "Sweet home";
    char *ptr = a_string;
    for (int i = 0; i < sizeof(a_string); i++){
        printf("%c", ptr[i]);
    }
}
```

Common Functions for Strings

[Prototype] toupper() and tolower()

https://cplusplus.com/reference/cctype/toupper/

prototype

```
int toupper ( int c );
```

https://cplusplus.com/reference/cctype/tolower/

prototype

```
int tolower ( int c );
```

toupper() and tolower()

C-course-materials/07-Strings-2/capitalize.c

C-course-materials/07-Strings-2/lower.c

toupper()

```c
#include <stdio.h>
#include <ctype.h>
int main(void){
    char str[] = "hello, world!";
    for (int i = 0; i < sizeof(str); i++){
        str[i] = toupper(str[i]);
    }
    printf("%s\n", str);
}
```

```
HELLO, WORLD!
```

tolower()

```c
#include <stdio.h>
#include <ctype.h>
int main(void){
    char str[] = "BELLO!";
    for (int i = 0; i < sizeof(str); i++){
        str[i] = tolower(str[i]);
    }
    printf("%s\n", str);
}
```

```
bello!
```

[Usage] `strcat` and `strcpy`

- strcat
  - string concatenation

```
char *strcat(char *destination, const char *source);
```

appends the contents of `source` to the end of the `destination`.

- strcpy
  - string copy

```
char *strcpy(char *destination, const char *source);
```

copy the contents of `source` to the `destination`.

Const

C-course-materials/07-Strings-2/const_example.c

- The qualifier (修飾子) `const` can be applied to the declaration of any variable to specify that its value will not be changed (we may change the value of the const variable by using a pointer).

```c
#include <stdio.h>
#include <ctype.h>
int main(void){
    // Error! We cannot modify a const variable.
    const char str[] = "hello, world!";
    for (int i = 0; i < sizeof(str); i++){
        str[i] = toupper(str[i]);
    }
    printf("%s\n", str);
}
```

Common Functions for Strings

System functions

- How to know what the C standard library includes? https://cplusplus.com/reference/
- Six common header files:

header file	Purpose	Examples
stdio.h	Standard I/O	printf(), scanf(), …
stdlib.h	Standard library (general tools)	abs(), malloc(), free(), exit(), …
math.h	math	sqrt(), pow(), sin(), cos(), ….
ctype.h	Functions related to characters	isalpha(), isdigit(), toupper(), …
string.h	Functions for processing strings	strcpy(), strlen(), strcmp(), …
time.h	time	time(), clock(), difftime(), …

Each header has its own topic!

[Prototype] fgets

prototype

```
char * fgets ( char * str, int num, FILE * stream );
```

- str
  - Pointer to an array of chars where the string read is copied.
- num
  - Maximum number of characters to be copied into str (**including the terminating null-character**).
- stream
  - Pointer to a FILE object that identifies an input stream.
  - stdin can be used as argument to read from the standard input.

fgets

C-course-materials/07-Strings-2/read_string.c

```c
#include <stdio.h>
#include <string.h>
int main(void){
    char str_to_load[100];
    printf("Enter a string (C Programming): ");
    // scanf("%s", str_to_load);
    fgets(str_to_load, 3, stdin);
    printf("You entered: %s\n", str_to_load);
}
```

Output

```
C
```

[Important Notes] fgets

- C does not check the boundary of a string.
- The `gets' function is dangerous and should not be used.

strlen: the function of string length

- `strlen` returns the length of a string, not including the null character.

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

int main(void){
    int len;
    len = strlen("abc"); /* len is now 3 */
    printf("%d\n", len);
    len = strlen(""); /* len is now 0 */
    printf("%d\n", len);

    char a_string[] = "Hello";
    printf("%d\n", strlen(a_string)); // 5
    printf("%d\n", sizeof(a_string)); // 6 (include \0)
}
```

[Important Notes] strlen vs. sizeof

- Given a string
  - strlen: Returns the length of the string **excluding** the null character (\0).
  - sizeof: Returns the total size of the object or variable in bytes. For a string declared as a character array, it **includes** the null character (\0).

[Prototype and Illustration] strcat

strcat

prototype

```
char * strcat ( char * destination, const char * source );
```

Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination.

destination

L	e	f	t	\0

source

R	i	g	h	t	\0

Result

L	e	f	t	R	i	g	h	t	\0

[Prototype and Illustration] strcpy

strcpy

prototype

```
char * strcpy ( char * destination, const char * source );
```

- Copies the C string pointed by source into the array pointed by destination, including the terminating null character (and stopping at that point).

destination

L	e	f	t	\0

source

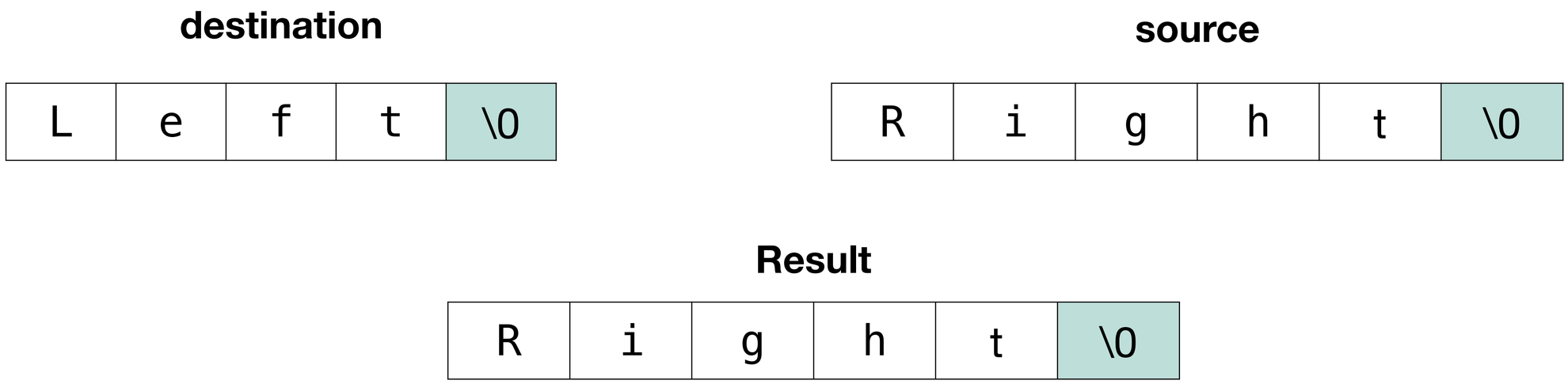
R	i	g	h	t	\0

Result

R	i	g	h	t	\0

[Important Notes] strcpy

- To avoid overflows, the size of the array pointed by ***destination*** shall be long enough to contain the same C string as *source* (including the terminating null character), and should not overlap in memory with *source*.



strcat vs. strcpy

C-course-materials/07-Strings-2/str_cat.c

C-course-materials/07-Strings-2/str_copy.c

- string concatenation (strcat)

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

int main(void){
    char str_1[12] = "Left";
    char str_2[] = " Right";
    strcat(str_1, str_2);
    printf("%s\n", str_1);
}
```

Output

```
Left Right
```

- string copy (strcpy)

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

int main(void){
    char str_1[7] = "Left";
    char str_2[] = " Right";
    strcpy(str_1, str_2);
    printf("%s\n", str_1);
    printf("%s\n", str_2);
}
```

Output

```
 Right
 Right
```

Problem of strcpy

C-course-materials/07-Strings-2/str_copy_problem.c

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

int main(void){
    char str_1[5] = "Left";
    char str_2[7] = " Right";
    strcpy(str_1, str_2);
    // printf("%s\n", str_1);
    printf("%s\n", str_2);
}
```

Let's assign the compact sizes of the two strings.

Output

```
t
```

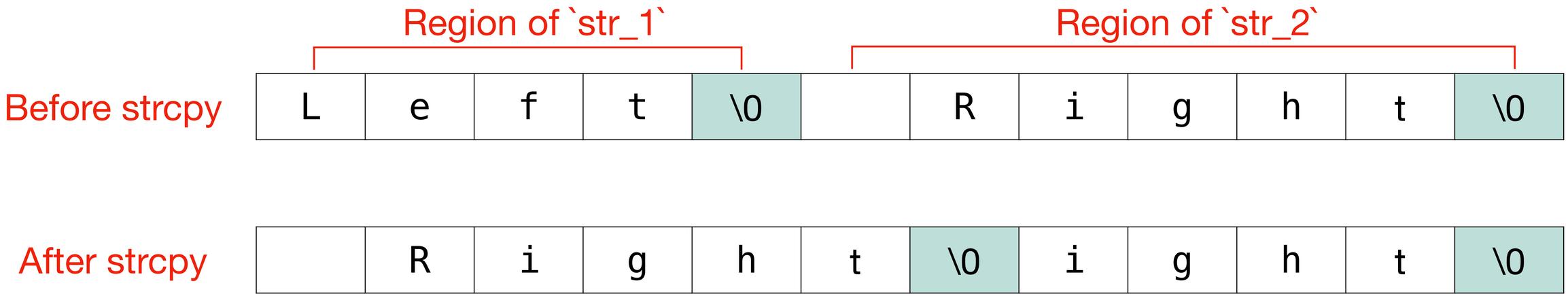
[Illustration] Problem of strcpy

```c
int main(void){
    char str_1[5] = "Left";
    char str_2[7] = " Right";
    printf("addr str_1: %p\n", str_1);
    printf("addr str_2: %p\n", str_2);
    strcpy(str_1, str_2);
    printf("str_1: %s\n", str_1);
    printf("str_2: %s\n", str_2);
}
```

C-course-materials/07-Strings-2/str_copy_problem.c

Output

```
t
addr str_1: 0x7fffffffdd7c
addr str_2: 0x7fffffffdd81
```



[Definition] Buffer overflow

- Buffer overflow (緩衝區溢位)
  - Or called “buffer overrun”
- An anomaly (不規則現象) whereby a program writes data to a buffer beyond the buffer's allocated memory, overwriting adjacent memory locations.
  - buffer: **a region of memory** used to store data temporarily

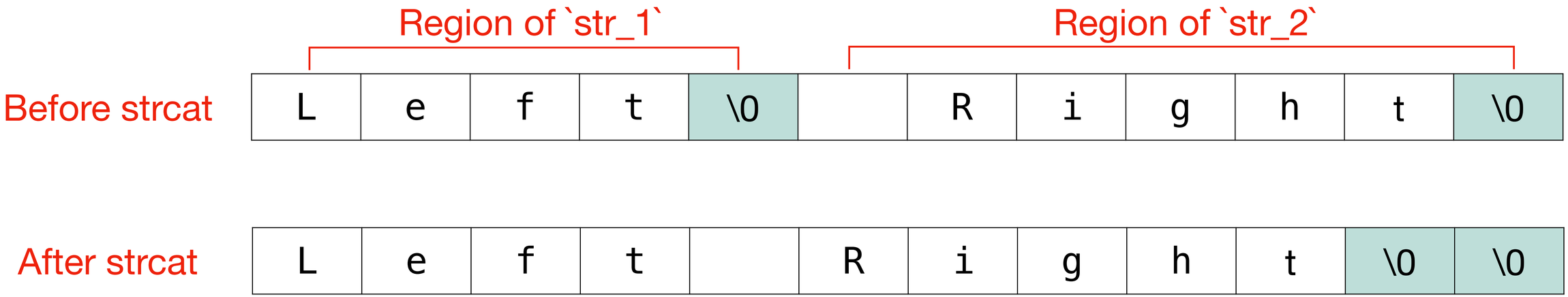
[Illustration] Problem of strcat

C-course-materials/07-Strings-2/str_copy_problem.c

```c
int main(void){
    char str_1[] = "Left";
    char str_2[] = " Right";
    printf("addr str_1: %p\n", str_1);
    printf("addr str_2: %p\n", str_2);
    strcat(str_1, str_2);
    printf("%s\n", str_1);
    printf("%s\n", str_2);
}
```

Output

```
t
addr str_1: 0x7fffffffdd7c
addr str_2: 0x7fffffffdd81
```



strncpy

C-course-materials/07-Strings-2/buffer_str_copy.c

prototype

```
char * strncpy ( char * destination, const char * source, size_t num );
```

```
#include <stdio.h>
#include <string.h>
int main(void){
    char str_1[5] = "Left";
    char str_2[] = " Right";
    strncpy(str_1, str_2, strlen(str_1));
    printf("%s\n", str_1);
    printf("%s\n", str_2);
}
```

```
 Rig
 Right
```

strncat

C-course-materials/07-Strings-2/buffer_str_cat.c

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

int main(void){
    char str_1[10] = "Left";
    char str_2[] = " Right";
    // sizeof(str_1): 10
    // strlen(str_2): 6
    strncat(str_1, str_2, sizeof(str_1) - 1 - strlen(str_2));
    printf("%s\n", str_1);
    printf("%s\n", str_2);
}
```

```
Left Ri
 Right
```

[Prototype] strcmp

- Compares the C string str1 to the C string str2.
- When the the contents of both strings are equal, `strcmp` returns 0.

prototype

```
int strcmp ( const char * str1, const char * str2 );
```

- When the the contents of both strings are not equal:

	ASCII values of the first unmatched characters in s1 and s2
return a positive integer	s1 > s2
return a negative integer	s1 < s2

https://cplusplus.com/reference/cstring/strcmp

strcmp (when s1 is equal to s2)

C-course-materials/07-Strings-2/str_cmp_equal.c

When the the contents of both strings are equal, `strcmp` returns 0.

```c
#include <stdio.h>
#include <string.h>
int main(void){
    char str_1[] = "hello world";
    char str_2[] = "hello world";
    printf("%d\n", strcmp(str_1, str_2));
}
```

Output

```
0
```

strcmp (when s1 is not equal to s2)

C-course-materials/07-Strings-2/str_cmp_neq.c

- strcmp returns a positive or a negative integer according to the ascii values of the first unmatched characters in the two strings when s1 is not equal to s2.

```c
#include <stdio.h>
#include <string.h>
int main(void){
    char str_1[] = "hello";
    char str_2[] = "bello";
    printf("%d\n", strcmp(str_1, str_2));
    printf("%d\n", strcmp(str_2, str_1));
}
```

Output

```
6
-6
```

	ASCII values of the first unmatched characters in s1 and s2
return positive	s1 > s2
return negative	s1 < s2