Week 4

XMUT-NWEN 241 - 2024 T2

Systems Programming

String

Mohammad Nekooei

School of Engineering and Computer Science

Victoria University of Wellington

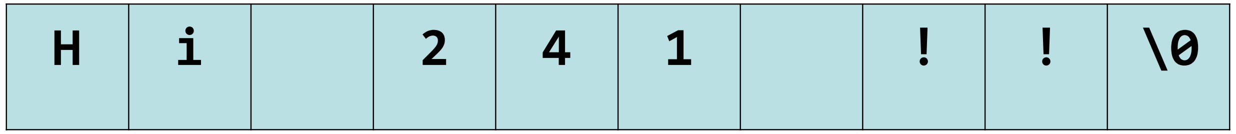
String

Content

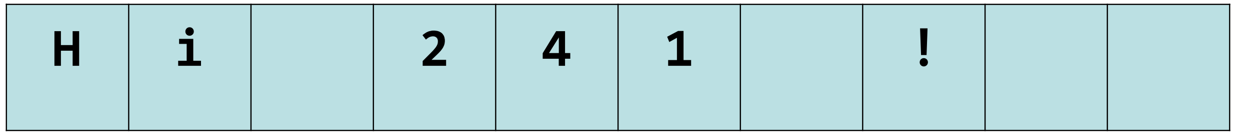
- Strings
- String Literal
- String Variable

What is String in C?

- C language **does not support strings** as a basic data type
- A C string is just an array that contains ASCII characters terminated by the **null character `'\0'`**
- A C string is stored in an array of chars



`"Hi 241 !!"`

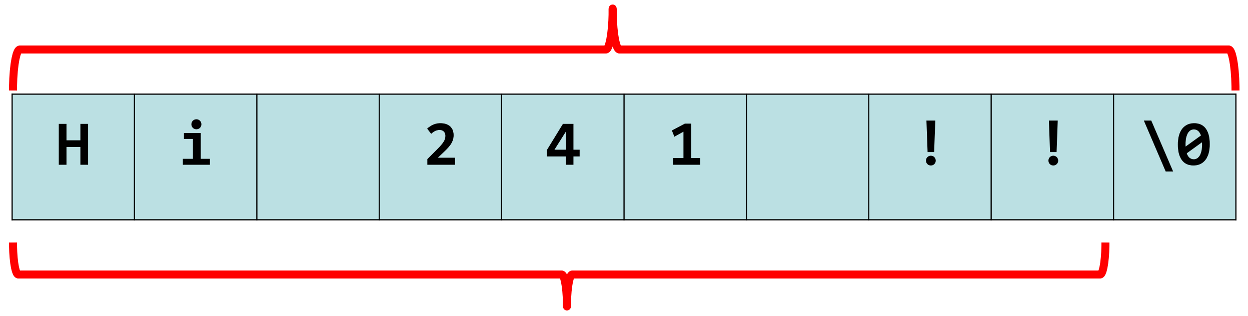


Not a valid string

String Length

- Number of bytes/characters **excluding** the null character

Entire string occupies 10 bytes /characters



String length = 9 bytes /characters

- `strlen()` function in `<string.h>` returns the string length

String Literal vs String Variable

- In C, we distinguish between **string literals** and **string variables**

- A **string literal** refers to the string constant value which is stored in the read-only memory area of the program

- A **string variable** refers to a string that is stored in an array which can be modified

String Literal (1)

- Enclosed in double quotes (") an can contain character literals (plain and escape characters)
- Can broken up into multiple lines (each line ends with \) or separated by whitespaces

```
"Hello, world"
```

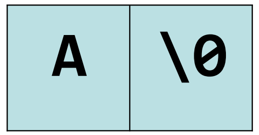
```
"Hello" ", " "world"
```

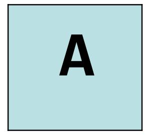
```
"Hello, \
world"
```

String Literal (2)

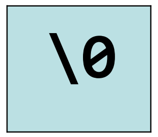
- String literals may contain as few as **one** or **even zero** characters

- Do not confuse a single-character string literal, e.g. `"A"` with a character constant, `'A'`
  - The former is actually two characters, because of the null-terminator stored at the end

“A” 

‘A’ 

- An **empty string**, `""`, consists of only the null-terminator, and is considered to have a string length of zero, because the null-terminator does not count when determining string lengths

`""` 

String Literal (3)

- String literals are passed to functions as *pointers* to a stored string. For example, given the statement:

```
printf("Hello world!\n");
```

  - The string literal `"Hello world!\n"` will be stored somewhere in memory, and the address will be passed to `printf()`
  - The first argument to `printf()` is actually defined as a `char *`

- We will revisit this when we talk about pointers

Operations on String Literals

- String literals may be subscripted

```
printf("%c\n", "Hello"[2]);    /* will print 'l' */
```

- Attempting to modify a string literal results in undefined behaviour, and may cause problems in different ways depending on the compiler, *e.g.*

~~`"Hello"[2] = 'e';`~~

Symbolic String Constants

- Similar to integer and float symbolic constants, symbolic string constants can be declared using **`const`** qualifier or **`#define`** pre-processor

```
const char *MSG = "Hello, world";
const char *MSG_A = "Hello, \
world";
const char *MSG_B = "Hello" ", " "world";
```

```
#define MSG   "Hello, world"
#define MSG_A "Hello, \
world"
#define MSG_B "Hello" ", " "world"
```

String Variables

- **String variables** are stored as arrays of chars, terminated by the **null character**
- A string variable can be initialized in 2 ways using the methods discussed in previous lecture:

```
char str[10];
str[0] = 'H';
str[1] = 'e';
str[2] = 'l';
str[3] = 'l';
str[4] = 'o';
str[5] = ' ';
str[6] = '!';
str[7] = '\0';
```

```
char str[10] = {'H', 'e', 'l',
     'l', 'o', ' ', '!', '\0' };
```

Efficient String Variable Initialization

- Another way to initialize a char array to hold a string variable: **assign a string literal to the array during declaration**

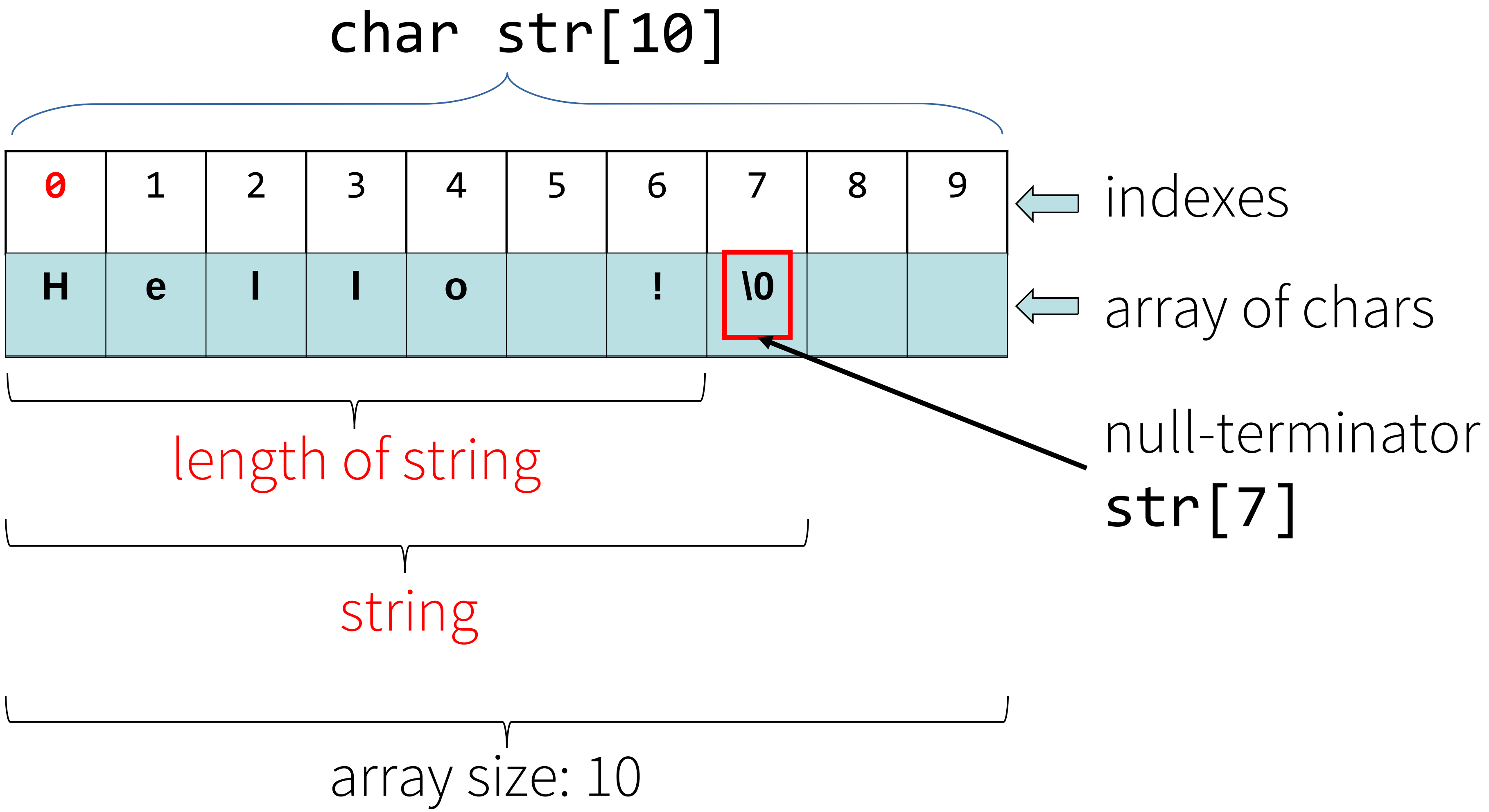
```
char str[10] = "Hello !";
```

```
char str[] = "Hello !";
```

What’s the difference between the two?

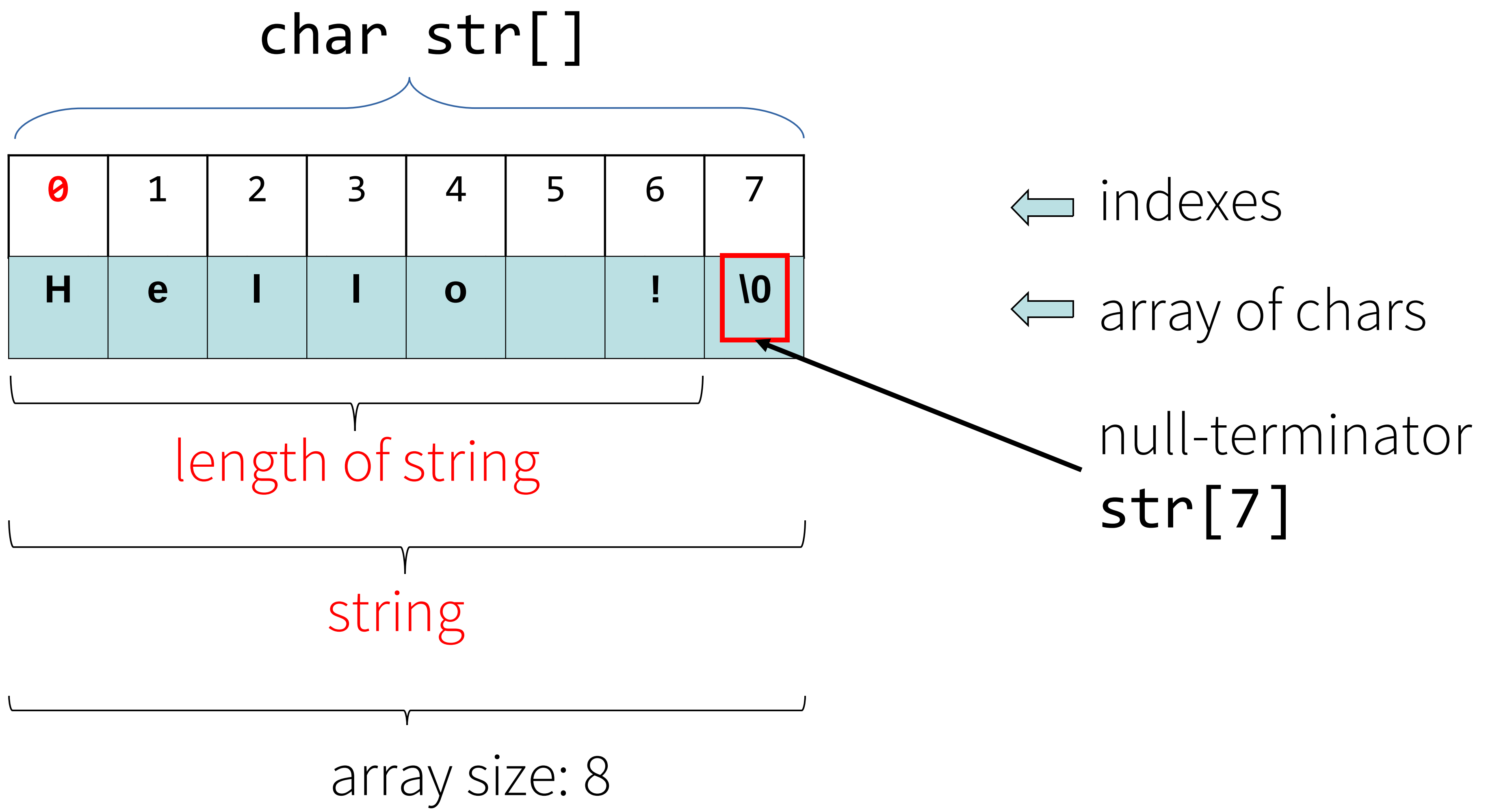
Arrays and C Strings

```
char str[10] = "Hello !";
```



Arrays and C Strings

```
char str[] = "Hello !";
```



Assigning a string after array declaration

```
char str[10];
…
str = "Hello !";
```

Illegal! Use `strcpy()` function

```
char str[10];
…
strcpy(str, "Hello !");
```