[SWE2015-41] Introduction to Data Structures (자료구조개론)

Strings

Department of Computer Science and Engineering

Instructor: Hankook Lee (이한국)

(Recap) What is an Array?

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**

Declaration in C

```
type name[size] = { ... };
```

main.c

```
int numbers[10] = {
    1, 5,  9, -3, 8,
    7, 6, 10, -5, 0
};
```

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	0x16aedf33c	10
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`

`numbers[2]`

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	0x16aedf33c	10
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`

`numbers[7]`

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	0x16aedf33c	**10**
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`
- Time complexity for the access = $O(1)$
  - Why?

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	**0x16aedf33c**	**10**
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`
- Time complexity for the access = $O(1)$
  - Address computation requires $O(1)$

```
numbers = &numbers[0] = 0x16aedf320
&numbers[7] = &numbers[0] + 7
            = 0x16aedf33c
```

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	**0x16aedf33c**	**10**
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`
- Time complexity for the access = $O(1)$
  - Address computation requires $O(1)$

```
numbers = &numbers[0] = 0x16aedf320
&numbers[i] = &numbers[0] + i
```

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	**0x16aedf33c**	**10**
8	0x16aedf340	-5
9	0x16aedf344	0

(Recap) Access Elements in Array

- An array is a collection of elements of **the same data type** in **a contiguous block of memory**
- The `i`-th element can be accessed by `arr[i]`
- Time complexity for the access = $O(1)$
  - Address computation requires $O(1)$
  - Value modification also requires $O(1)$

`numbers[7] = 2`

Index	Address	Value
0	0x16aedf320	1
1	0x16aedf324	5
2	0x16aedf328	9
3	0x16aedf32c	-3
4	0x16aedf330	8
5	0x16aedf334	7
6	0x16aedf338	6
7	**0x16aedf33c**	**2**
8	0x16aedf340	-5
9	0x16aedf344	0

What is a String?

- A string is **an array of characters**
  - You can initialize its value by `"..."`

```
char msg[20] = "Hello, world!";
```

  - You can print the string value using the `%s` type specifier

```
printf("%s", msg);
```

What is a String?

- A string is **an array of characters**
  - You can initialize its value by `"..."`

```
char msg[20] = "Hello, world!";
```

  - You can print the string value using the `%s` type specifier

```
printf("%s", msg);
```

  - The null character `\0` represents the end of a string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	e	l	l	o	,		w	o	r	l	d	!	\0						

What is a String?

- A string is **an array of characters**
  - You can initialize its value by `"..."`

```
char msg[20] = "Hello, world!\0ABCD";
```

  - You can print the string value using the `%s` type specifier

```
printf("%s", msg); // Printed message: Hello, world!
```

  - The null character `\0` represents the end of a string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	e	l	l	o	,		w	o	r	l	d	!	\0	A	B	C	D	\0	

    - In this string, `"ABCD\0"` are ignored

Length of A String

- How to compute the length of a string?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	e	l	l	o	,		w	o	r	l	d	!	\0						

Length = 13

The end of this string

  - This is equivalent to finding the first null character

```
int strlen(char *str) {



}
```

Length of A String

- How to compute the length of a string?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	e	l	l	o	,		w	o	r	l	d	!	\0						

Length = 13

The end of this string

  - This is equivalent to finding the first null character

```
int strlen(char *str) {
    for (int i = 0; ; i ++)
        if (*(str+i) == '\0')
            return i;
}
```

Reverse A String

- How to reverse the order of a string in-place?

0	1	2	3	4	5	6
H	e	l	l	o	!	\0

Reverse order →

0	1	2	3	4	5	6
!	o	l	l	e	H	\0

Reverse A String

- How to reverse the order of a string in-place?

0	1	2	3	4	5	6
H	e	l	l	o	!	\0

Reverse order →

0	1	2	3	4	5	6
!	o	l	l	e	H	\0

  - For a 6-length string, you need to …
    - Swap `str[0]` and `str[5]` characters
    - Swap `str[1]` and `str[4]` characters
    - Swap `str[2]` and `str[3]` characters
  - For a `n`-length string, how to … ?

Reverse A String

- How to reverse the order of a string in-place?

0	1	2	3	4	5	6
H	e	l	l	o	!	\0

Reverse order →

0	1	2	3	4	5	6
!	o	l	l	e	H	\0

  - For a 6-length string, you need to …
    - Swap `str[0]` and `str[5]` characters
    - Swap `str[1]` and `str[4]` characters
    - Swap `str[2]` and `str[3]` characters
  - For a `n`-length string, how to … ?
    - Swap `str[i]` and `str[n-i-1]` characters

Reverse A String

- How to reverse the order of a string in-place?

0	1	2	3	4	5	6
H	e	l	l	o	!	\0

Reverse order →

0	1	2	3	4	5	6
!	o	l	l	e	H	\0

```
void reverse(char *str) {
    int len = strlen(str); // Compute the length of str
    char temp;
    for (int i = 0; i < len/2; i ++) {
        // Swap i-th and (len-i-1)-th elements
        temp = str[i];
        str[i] = str[len-i-1];
        str[len-i-1] = temp;
    }
}
```

Coding Practices

- Append a string `a[]` to another string `b[]`
  - Example: `a[] = "hello" & b[] = ", world"` → `b[] = "hello, world"`

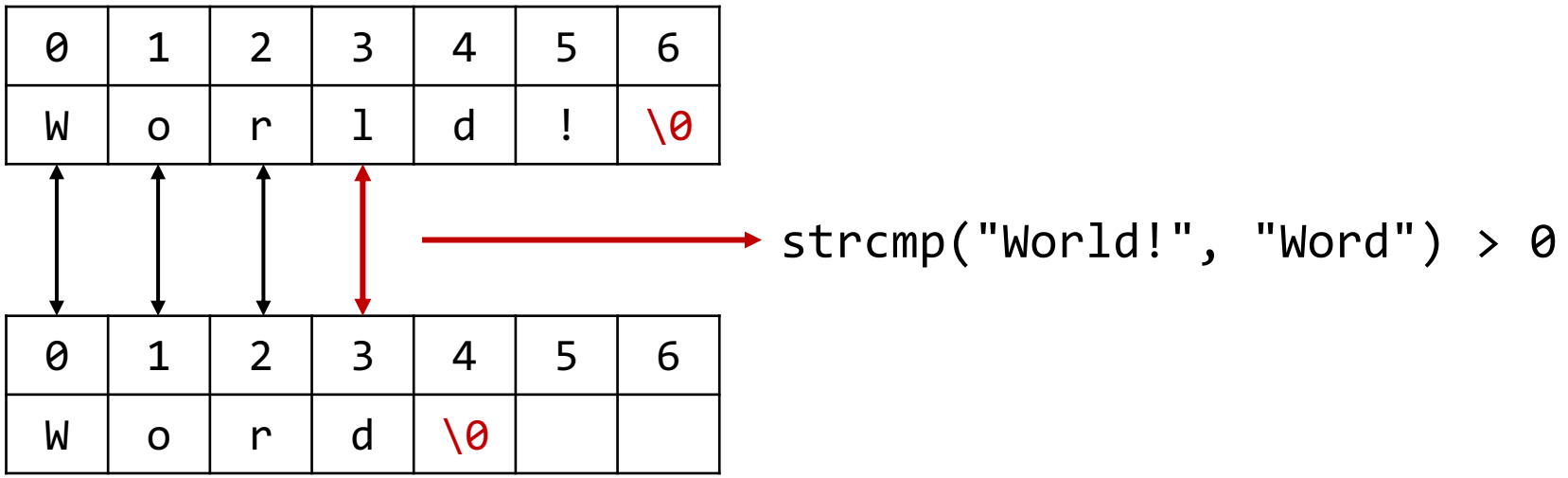
```
void strcat(char *a, char *b) {
    int alen = strlen(a), blen = strlen(b); // Compute their lengths
    // Append a to b
}
```

- Check whether a string `a[]` is the prefix of another string `b[]`
  - Example: `a[] = "he" & b[] = "hello"` → `return value = true`
  - Example: `a[] = "eh" & b[] = "hello"` → `return value = false`

```
bool isPrefix(char *a, char *b) {
    int alen = strlen(a), blen = strlen(b); // Compute their lengths
    // Check whether a is prefix of b
}
```

Compare Two Strings

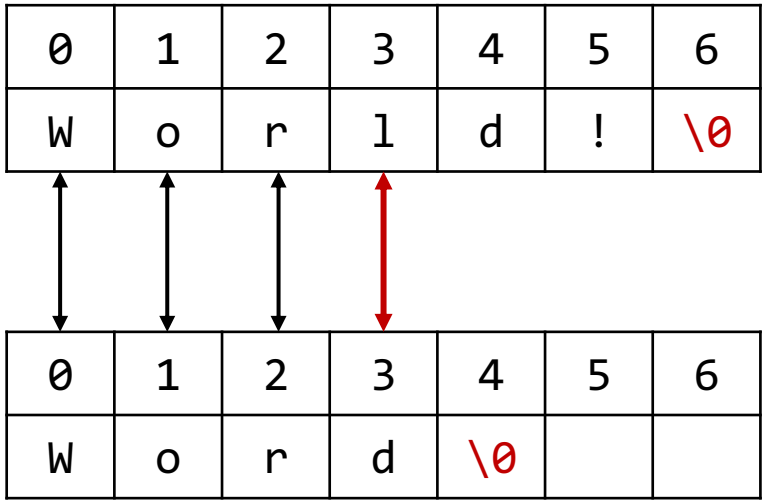
- Comparison between two strings `a[]` and `b[]`
  - `strcmp(a, b) == 0` : two strings are equal
  - `strcmp(a, b) < 0` : the string `a` comes before the string `b` in dictionary order
  - `strcmp(a, b) > 0` : the string `a` comes after the string `b` in dictionary order



Compare Two Strings

- Comparison between two strings `a[]` and `b[]`
  - `strcmp(a, b) == 0` : two strings are equal
  - `strcmp(a, b) < 0` : the string `a` comes before the string `b` in dictionary order
  - `strcmp(a, b) > 0` : the string `a` comes after the string `b` in dictionary order

```
int strcmp(char *a, char *b) {
    while (*a != 0 || *b != 0) {
        // Compare characters one by one
        if (*a > *b) return 1;
        if (*a < *b) return -1;
        // If they are same, move forward
        a += 1;
        b += 1;
    }
    return 0;
}
```



Pattern Matching

- What is the pattern matching problem?

  String: Welcome to the world of programming

  Pattern: world

  - Find the position in **the string** where **the pattern** first occurs

- A naive approach: $O(NM)$ where N & M are lengths of the string & the pattern

```
int strstr(char *str, char *pattern) {
    int N = strlen(str), M = strlen(pattern); // Compute their lengths




    return -1;
}
```

Pattern Matching

- What is the pattern matching problem?

  String: Welcome to the world of programming

  Pattern: world

  - Find the position in **the string** where **the pattern** first occurs

- A naive approach: $O(NM)$ where N & M are lengths of the string & the pattern

```
int strstr(char *str, char *pattern) {
    int N = strlen(str), M = strlen(pattern); // Compute their lengths
    for (int i = 0; i <= N-M; i ++) { // Check main[i, ..., i+M-1] == pattern[0, ..., M-1]
        for (int j = 0; j < M; j ++) if (main[i+j] != pattern[j]) break;
        if (j == M) return i;
    }
    return -1;
}
```

Pattern Matching

- What is the pattern matching problem?

```
String: Welcome to the world of programming
Pattern: world
```

  - Find the position in **the string** where **the pattern** first occurs

- A naive approach: $O(NM)$ where N & M are lengths of the string & the pattern

- The best approach: KMP algorithm, $O(N + M)$
  - https://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm
  - https://youtu.be/pu2aO_3R118
  - **Note.** This algorithm is not covered by this course

Coding Practices

- Check a string `a[]` is a palindrome
  - Example: `a[] = "tenet"` is a palindrome
  - Example: `a[] = "radar"` is a palindrome
  - Example: `a[] = "hello"` is not a palindrome

```
int isPalindrome(char *a) {
    // return 1 if a is palindrome
}
```

- Check a string `a[]` consists of digits only
  - Example: `a[] = "1102131"` is a digit string
  - Example: `a[] = "+423"` is not a digit string

```
int isDigit(char *a) {
    // return 1 if all characters in the string are digits
}
```

String Operations in `<string.h>`

- There are many pre-defined string operations in `<string.h>`

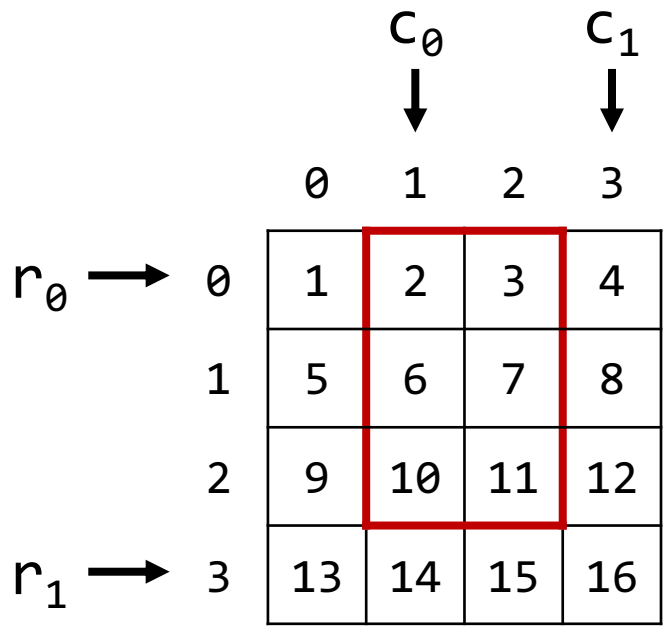
```
#include <string.h>
```

  - `strcpy(char *dest, char *source)` - copy a string to a certain memory
  - `strcat(char *dest, char *source)` - concatenate two strings
  - `strcmp(char *str1, char *str2)` - compare two strings
  - `strlen(char *str)` - compute the length of a string
  - `strstr(char *str1, char *str2)` - find the position where `str2` first occurs
  - …
  - https://www.tutorialspoint.com/c_standard_library/string_h.htm

Programming Assignment #1

- **Deadline:** ~ 04/02 23:59:59
- **(Q)** Given a matrix, can you compute the sub-matrix sum efficiently?
  - Sub-matrix is defined by row and column indices (r_0, r_1, c_0, c_1)

$$A[i, j],\ r0 \le i < r1,\ c0 \le j < c1$$



sum = 2+3+6+7+10+11 = 39

Any Questions?