Advanced
C Programming & Lab

10. Strings

Sejong University

Outline

1) **Strings?**

2) **Strings and Pointers**

3) **Arrays of Strings**

4) **Strings and Functions**

5) **Strings and Character Input/Output**

1) Strings

- **Arrays of Characters**
  - Elements of an array are characters
    - ✓ Process each element: Initialization, Input/Output

```
char str[8] = {'H','e','l','l','o'}; // Initialization
int i;

for (i=0 ; i<5 ; i++)
  printf("%c", str[i]);  // Character Output
```

  - Easier to process characters as a whole

    ➔ **Strings**

1) Strings

- **Strings: A sequence of characters**
- **Expression?**
  - Double quotes
  - Ex)
    - ✓ "Hello" , "A" , "123"
    - ✓ Caution: 'A' **is a character**

- **How to Store?**
    - ✓ Stored as an array of characters

- **Input/Output?**
  - scanf, printf: conversion specification, %s

1) Strings

- **Store and Initialize Strings**
  - As an array of characters
  - Initialization: double quotes

  - Example

```
char str[8] = "Hello";  // specify the size of an array

char str[ ] = "Hello";  // do not specify the size of an array
                        // determined by the initialization
```

1) Strings

- **Input/Output: scanf, printf**
  - Process a whole string
  - Conversion specification %s
  - Give the starting address of a string

```
char str[8];

scanf("%s", str);      // String Input

printf("%s", str);     // String Output
```

  - scanf: do not need & symbol
  - printf: give the starting address of an array when using %s

1) Strings

- **An array of characters to a string?**

```
char str[8] = {'H','e','l','l','o'}; // Initialization
int i;

for (i=0 ; i<5 ; i++)
  printf("%c", str[i]);  // Character Output
```

```
char str[8] = "Hello";  // Strings Initialization

printf("%s", str);     // String Output
```

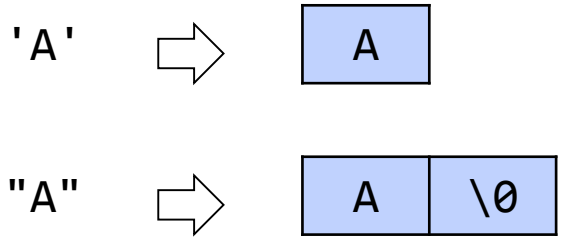
1) Strings

- **[Practice 1] Write a program**
  - Declare a character array *str* of size 10
  - Initialize the above array to “Hello” as declared
  - Print out the string *str*
  - Read a string “World” from a user and store it in *str*
  - Print out the string *str*

1) Strings

- **null character `'\0'`**
  - Indicate the end of a string
  - ASCII value: 0, i.e., `'\0' == 0`
  - Strings always contain a null character at the end

  - Difference between a character 'A' and string "A"



1) Strings

- **null character**
  - Ex) Initialization

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

||

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

str	H	e	l	l	o	\0
	[0]	[1]	[2]	[3]	[4]	[5]

1) Strings

- **Initialization: Characters vs Strings**
  - Size comparison

```
char str1[] = {'H','e','l','l','o'}; // As a character
char str2[] = "Hello";  // As a string

printf("%d %d", sizeof(str1), sizeof(str2) );
Result:
5 6
```

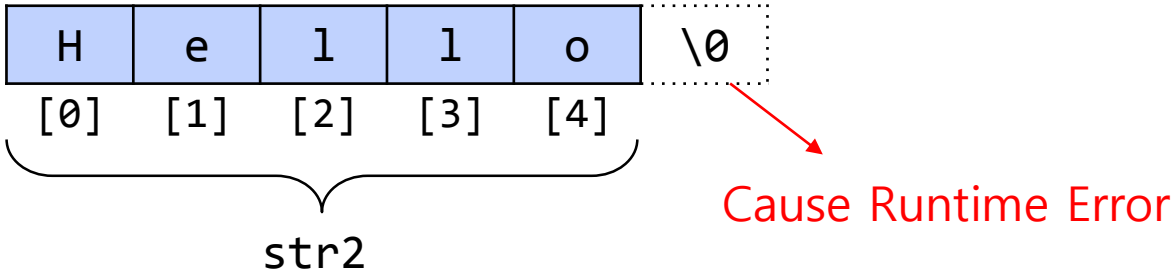
str1	H	e	l	l	o
	[0]	[1]	[2]	[3]	[4]

str2	H	e	l	l	o	\0
	[0]	[1]	[2]	[3]	[4]	[5]

1) Strings

- **Size of a character array**
  - The number of characters + 1

```
char str1[6] = "Hello";       // OK
char str2[5] = "Hello";       // Runtime Error
```



1) Strings

- **String Output: printf**
  - Use conversion specification: %s
  - Print strings referred by the given address

```
char str[20] = "Hello World";
int i;
for( i=0 ; i < 20 ; ++i )
   printf("%c", str[i]);
printf("..\n");

Result:
Hello World          ..
```

Add extra null characters
Print '\0‘ as an empty space

```
char str[20] = "Hello World";

printf("%s..\n", str);

Result :
Hello World..
```

Print 11 characters, but the size of an array is 20

1) Strings

- **printf: %s and null character**
  - Print from the character referred by the given address to a null character
    - ✓ Do not consider the size of an array
    - ✓ printf does not know the size of an array

```
char str[20] = "Hello World";
int i;
for( i=0 ; i < 20 ; ++i )
   printf("%c", str[i]);
printf("..\n");

Result:
Hello World          ..
```

```
char str[20] = "Hello World";

printf("%s..\n", str);
printf("%s..\n", str+5);

Result :
Hello World..
 World..
```

1) Strings

- **printf: %s and null character**
  - Use %c instead of %s?

```
printf("%s..\n", str);
```

||

```
for( i=0 ; str[i] ; ++i )
   printf("%c", str[i]);
```

```
As long as the
element of an array
is true

Not 0
Not '\0'
```

1) Strings

- **printf: %s and null character**

```
char str[20] = "Hello World";
int i;
str[7] = '\0';
for( i=0 ; i < 20 ; ++i )
   printf("%c", str[i]);
printf("..\n");
```

Result:

```
Hello W rld          ..
```

```
char str[20] = "Hello World";

str[7] = '\0';
printf("%s..\n", str);
printf("%s..\n", str+5);
```

Result :

```
Hello W..
 W..
```

1) Strings

- **[Practice 2] Write a program**
  - Declare a character array of size **6**
  - Read a string "Hello" from a user and store it in *str*
  - Print out the string *str*
  - Insert '?' into *str*[5]
  - Print out the string *str*
    - ✓ What happened?

1) Strings

- **scanf: %s and null character**
  - Read a string from a user
  - Put a null character at the end

```
char str[20];

scanf("%s", str);
printf("%s..\n", str);
scanf("%s", str+5);
printf("%s..\n", str);
```

Example

```
Hello        → Input
Hello..      → Output
World        → Input
HelloWorld.. → Output
```

1) Strings

- **scanf: %s and null character**
  - %s: characters before a new-line, white space, tab character to be considered as a string

```
char str[20];

scanf("%s", str);
printf("%s..\n", str);
```

Example

```
Hello World          → Input
Hello..              → Output
```

1) Strings

- **Caution 1**
  - Can initialize a string to a character array
  - Not vice versa

```
char str[20];

str[0] = 'a';   // OK
str = "Hello World";  // Compilation Error
```

✓ Change the starting address of str

- There is a function to copy a character array into a string
  → will cover later

1) Strings

- **Caution 2**
  - Should have enough spaces to store strings

```
char str[5];

scanf("%s",str);
...
```

  - If enter "HelloWorld"?
    - ✓ Exceed the size of the array of size 5 ➔ Runtime Error
    - ✓ The size of the array should be 11 or larger (Why?)

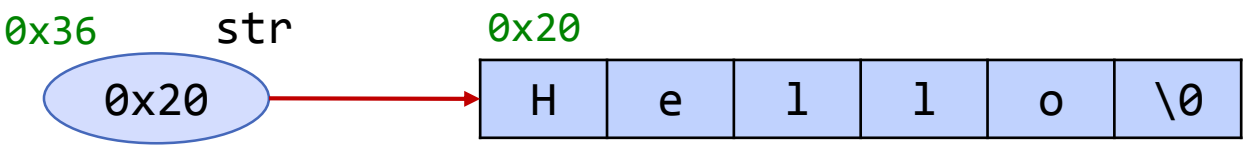
Outline

1) Strings
2) **Strings and Pointers**
3) Arrays of Strings
4) Strings and Functions
5) Strings and Character Input/Output

2) Strings and Pointers

- **String pointer**
  - A pointer points a string
    - ✓ Declare a pointer str, pointing a string "Hello"
    - ✓ str contains an address

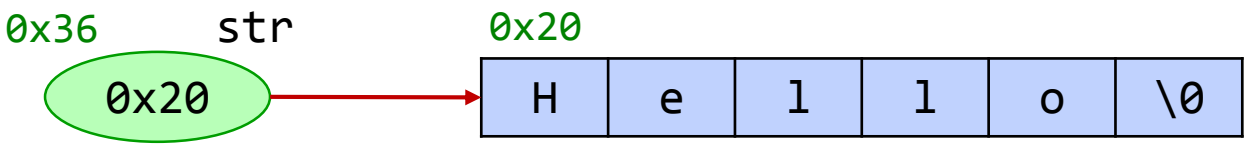
```
char *str = "Hello";

printf("%s..\n", str);
```



2) Strings and Pointers

- **Use a string pointer like an array**

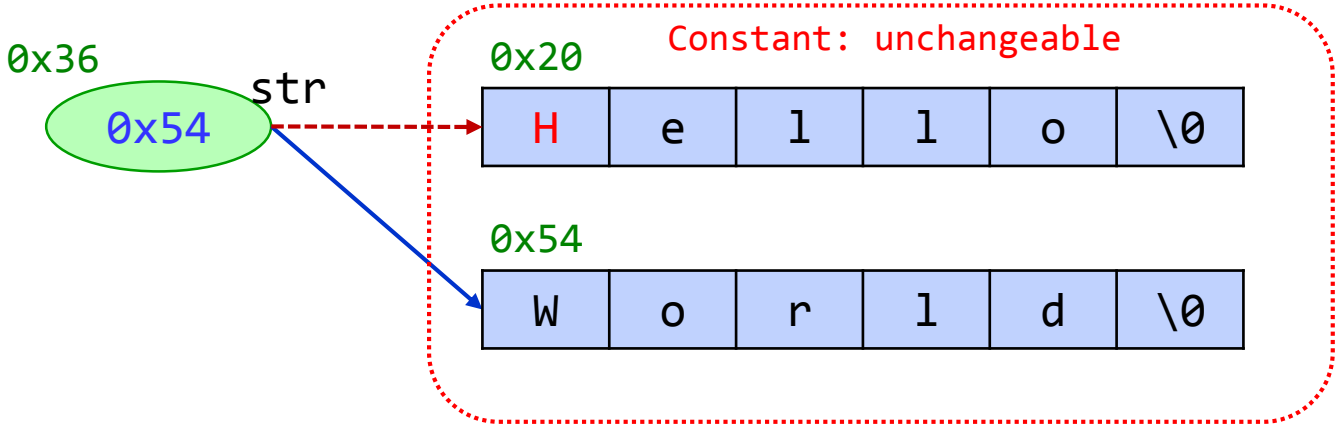
```
char *str = "Hello";

for (i=0 ; i<5 ; i++)
  printf("%c", str[i]);  // Print characters
```



2) Strings and Pointers

- **Properties**
  - "Hello" is a constant string, **Cannot** modify it
  - But, str is a variable, can modify it

```
char *str = "Hello";

str[0] = 'h';  // Cannot change (Runtime Error)
str = "World";  // OK to change str
```

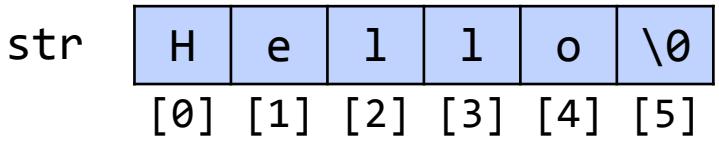


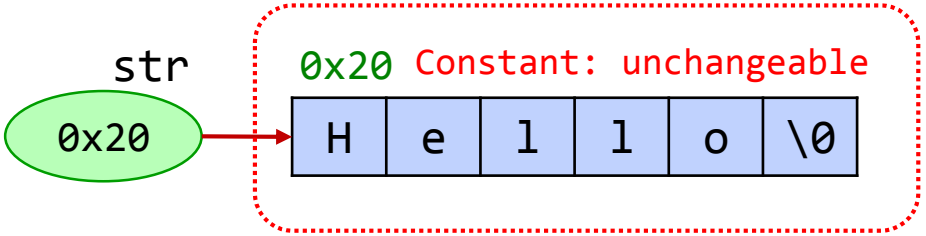
2) Strings and Pointers

- **Comparison: Character arrays vs String pointers**

```c
char str[6] = "Hello";

printf("%c", str[0]); // O
printf("%s", str);    // O

str[0] = 'h';         // O
scanf("%s", str);     // O

str = "World";        // X
```

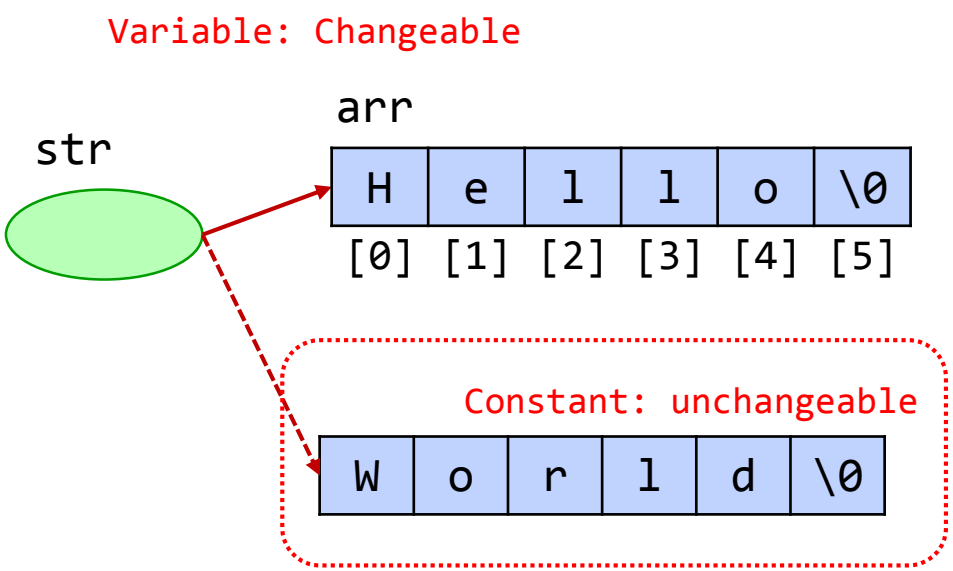
```c
char *str = "Hello";

printf("%c", str[0]); // O
printf("%s", str);    // O

str[0] = 'h';         // X
scanf("%s", str);     // X

str = "World";        // O
```





2) Strings and Pointers

- **Caution!!**
  - Not because str is a pointer
  - **But because str points to something unchangeable**




```
char arr = "Hello";
char *str = arr;

printf("%c", str[0]); // O
printf("%s", str);    // O

str[0] = 'h';         // O
scanf("%s", str);     // O

str = "World";        // O

str[0] = 'w';         // X
scanf("%s", str);     // X
```

2) Strings and Pointers

- **[Practice 3] Write a program**
  - Declare a string pointer pc (variable),
    Initialize it to “To be, or not to be : that is the question"
  - Use a loop, print how many times a lower-case letter 't' appears

Outline

1) Strings

2) Strings and Pointers

3) **Arrays of Strings**

4) Strings and Functions

5) Strings and Character Input/Output

3) Arrays of Strings

- **Multiple strings: Multiple character arrays**

```
char num0[5] = "zero";
char num1[5] = "one";
char num2[5] = "two";
printf("%s\n", num0);
printf("%s\n", num1);
printf("%s\n", num2);
```

	[0]	[1]	[2]	[3]	[4]
num0	z	e	r	o	\0
num1	o	n	e	\0	
num2	t	w	o	\0	

3) Arrays of Strings

- **Multiple strings: An array of strings**
  - Use 2-D character arrays
    - ✓ Data type of num[0], num[1], num[2] is char *

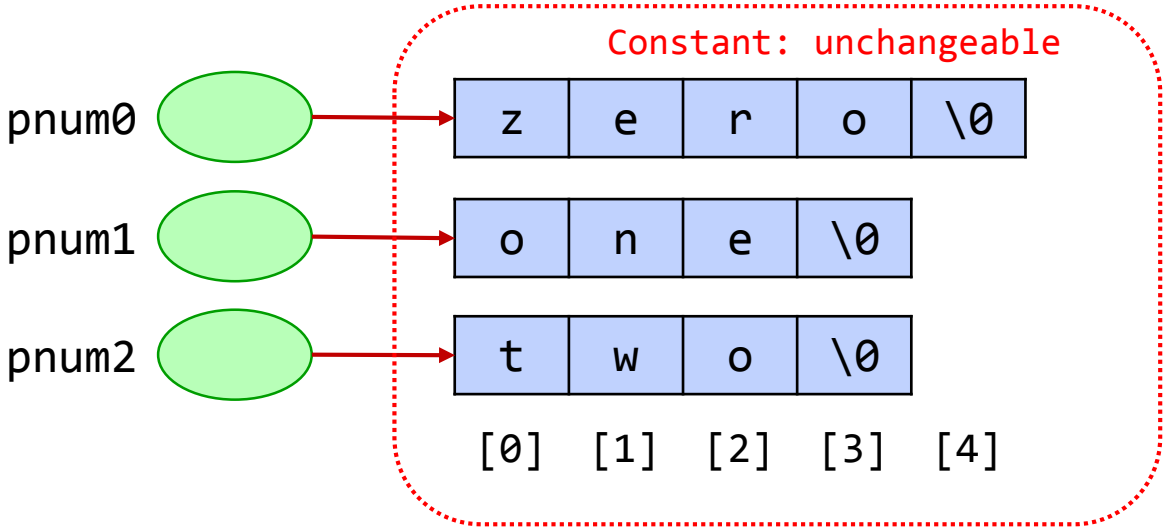
```
char num[3][5] = {"zero", "one", "two"};
int i;
for( i=0; i < 3; ++i ) printf("%s\n", num[i]);
```

	[0]	[1]	[2]	[3]	[4]
num[0]	z	e	r	o	\0
num[1]	o	n	e	\0	
num[2]	t	w	o	\0	

3) Arrays of Strings

- **Multiple strings: Multiple string pointers**

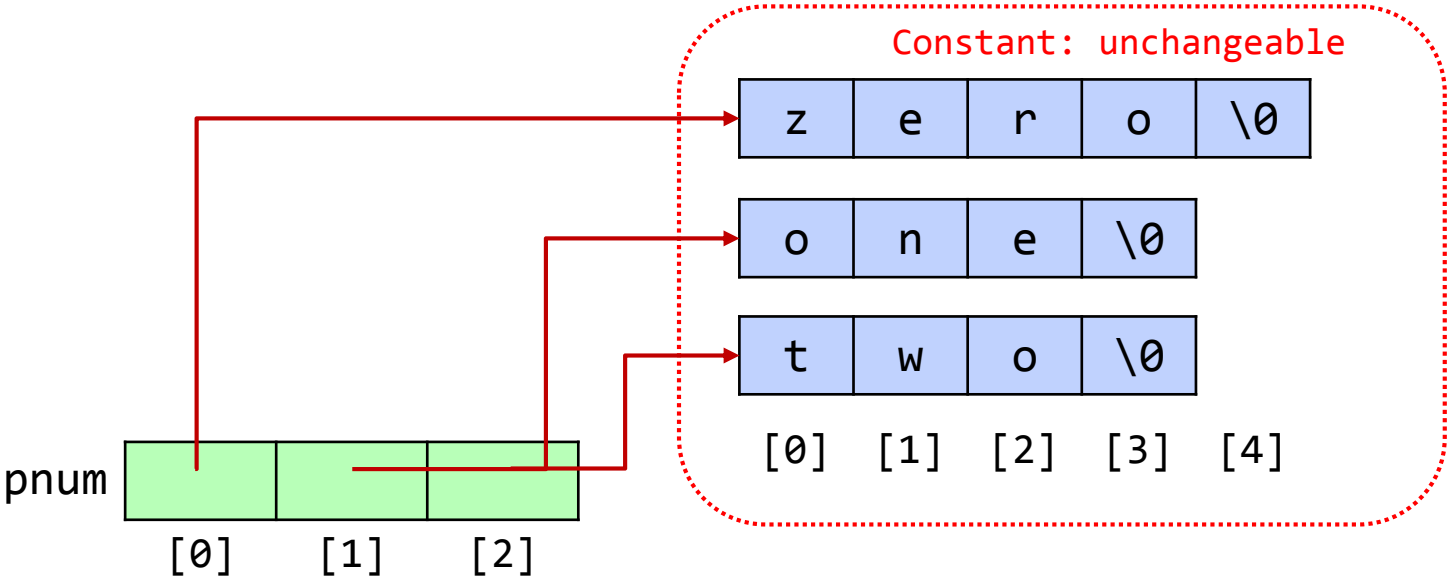
```
char *pnum0 = "zero";
char *pnum1 = "one";
char *pnum2 = "two";
printf("%s\n", pnum0);
printf("%s\n", pnum1);
printf("%s\n", pnum2);
```



3) Arrays of Strings

- **Multiple strings: String pointers**
  - Array of character pointers

```
char *pnum[3] = {"zero", "one", "two"};
int i;
for( i=0; i < 3; ++i ) printf("%s\n", pnum[i]);
```



3) Arrays of Strings

- **[Practice 4] Write a program**
  - Declare a 2-D character array of size 3x20
  - Initialize it to
    - ✓ "Time is gold"
    - ✓ "No pain no gain"
    - ✓ "No sweat no sweet"
  - Use a nested loop, print how many times a lower-case letter 'a' appears
  - (Extra) Use a character pointer instead of 2-D character array

Outline

1) Strings

2) Strings and Pointers

3) Arrays of Strings

4) **Strings and Functions**

5) Strings and Character Input/Output

4) Strings and Functions

- **Calculate the length of a string**
  - Use a null character and loop

```
char str[20] = "Hello World";
int i;

for( i=0; str[i] != '\0' ; ++i ) ;

printf("length: %d\n", i);
Result:
length: 11
```

4) Strings and Functions

- **Calculate the length of a string (Use standard functions)**
  - **strlen** function
    - ✓ `unsigned int strlen(char *s)`
    - ✓ Return the length of a string s

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

void main(){
   char str[20] = "Hello World";
   printf("length: %d\n", strlen(str));
}
Result:
length: 11
```

4) Strings and Functions

- **Standard functions: Strings**
  - C provides a variety of standard functions
  - Header file <string.h> contains them
    - ✓ Use include like:

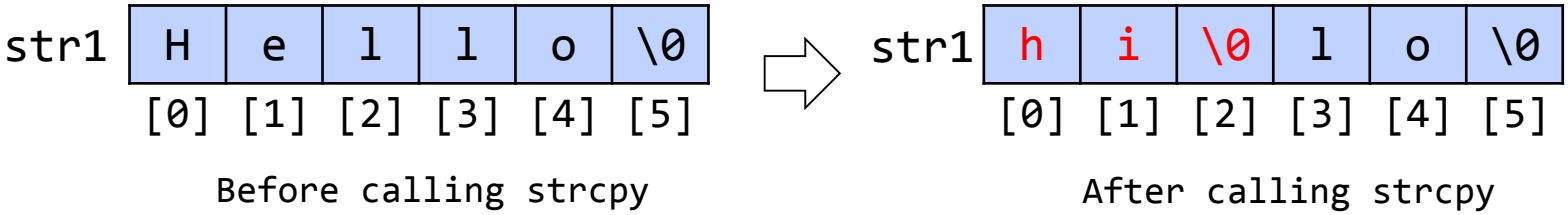
      #include <stdio.h>

  - Convenient to use the standard functions
  - Need to know how to use them

4) Strings and Functions

- `char *strcpy(char *s1, char *s2)`
  - copy a string s2 to s1
    does not change s2

```
char str1[6] = "Hello";

strcpy( str1, "hi");

printf("str1: %s..\n", str1);
Result:
str1: hi..
```



4) Strings and Functions

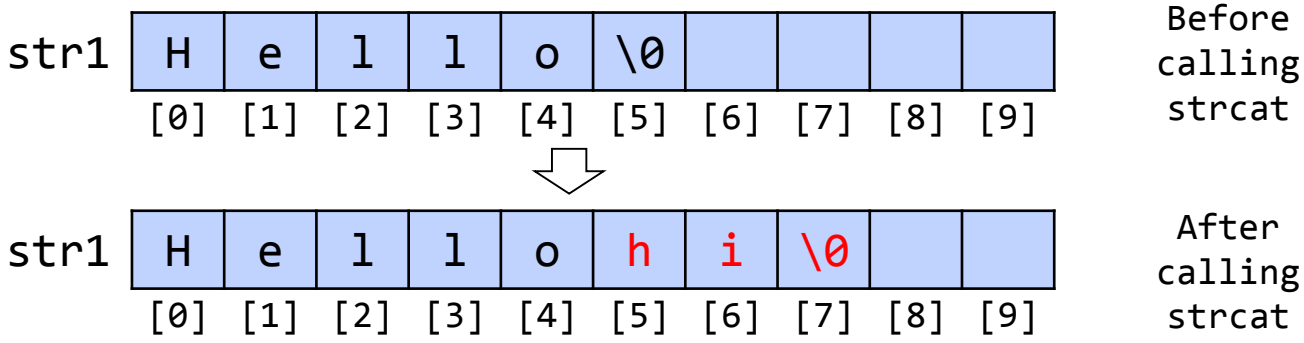
- **Caution: strcpy(s1, s2)**
  - The size of s1 should be at least the size of s2 + 1
    - ✓ Otherwise, Runtime Error

```
char s1[10], s2[5] = "hi";
char *s3 = NULL;

strcpy( s1, s2);        // OK

strcpy( s2, "Hello");   // Runtime Error
strcpy( s3, "Hello");   // Runtime Error

s3 = s1;
strcpy( s3, "Hello");   // OK
```

4) Strings and Functions

- `char *strcat(char *s1, char *s2)`
  - concatenate a string s1 and a sting s2
    does not change s2

```
char str1[10] = "Hello";

strcat( str1, "hi");

printf("str1: %s..\n", str1);
Result:
str1: hellohi..
```



4) Strings and Functions

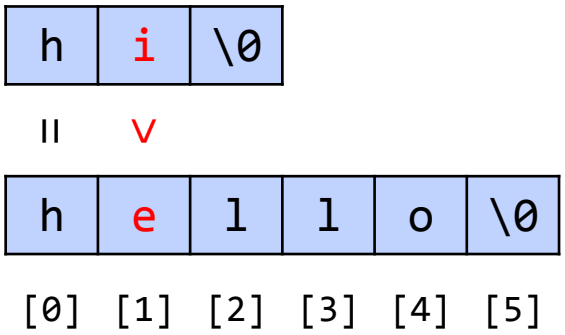
- **Caution: strcat(s1, s2)**
  - s1 should be large enough to hold the concatenated string
    - ✓ Otherwise, Runtime Error

```
char s1[10] = "Hello", s2[5] = "hi", s3[20];
char *s4 = NULL;

strcat( s2, s1);      // Runtime Error
strcat( s3, s1);      // Runtime Error (why?)
strcat( s4, s1);      // Runtime Error
```

4) Strings and Functions

- `int *strcmp(char *s1, char *s2)`
  - Compare s1 and s2 (lexicographical)
    if s1>s2, positive number. if s1 < s2, negative number
    if s1 == s2, 0.
  - Compare each character
    - ✓ Use ASCII

```
printf("%d\n", strcmp("hi", "hello") );
Result:
1
```



4) Strings and Functions

- **Example**

```
char *str = "hi";

printf("%d\n", strcmp(str, "hi") );
printf("%d\n", strcmp(str, "Hi") );
printf("%d\n", strcmp(str, "hi~") );
printf("%d\n", strcmp(str, str) );
printf("%d\n", strcmp("hi", "high") );
printf("%d\n", strcmp("hi", ".") );
```

Result

```
0
1
-1
0
-1
1
```

4) Strings and Functions

- **[Practice 5] Write a program**
  - Read two strings A and B
  - Length of A and B is (no more than) 20, no white space, tab, new-line
  - Two strings are not identical

1) Print the length of strings A and B
2) Print the string that appears first in lexicographical order
3) Create a new string C containing ABA and print it out

Input

```
welcome
helloworld!!
```

Output

```
7 12
helloworld!!
welcomehelloworld!!welcome
```

4) Strings and Functions

- **Convert to a decimal number**
  - int atoi(char *s) : int type
  - long atol(char *s) : long type
  - double atof(char *s) : double type
  - Available in <stdlib.h>

```
printf("%d\n", atoi("123") );
printf("%d\n", atoi("-123") );

printf("%f\n", atof("-123") );
printf("%f\n", atof("123.45") );
```

Result

```
123
-123

-123.000000
123.450000
```

4) Strings and Functions

- **Functions**
  - Data type of s, s1, s2: char *

Function Prototype	Description
unsigned int **strlen**(s)	Length of a string s
char ***strcpy**(s1, s2)	Copy a string s1 into s2
char ***strcat**(s1, s2)	Concatenate strings s1 and s2
int ***strcmp**(s1, s2)	Compare strings s1 and s2 (lexicographical order)
int **atoi**(s)	Convert a string to int type, long type, double type Ex) atoi("12"), an integer 12
long **atol**(s)	
double **atof**(s)	

Outline

1) Strings

2) Strings and Pointers

3) Arrays of Strings

4) Strings and Functions

5) **Strings and Character Input/Output**

5) Strings and Character Input/Output

- **Input/Output Function**
  - printf and scanf : General-purpose function
    - ✓ Heavy and slow
  - C provides Input/Output functions for strings and characters
    - ✓ Fast
    - ✓ String input/output: puts, gets
    - ✓ Character input/output: putchar, getchar
  - Available in <stdio.h>

5) Strings and Character Input/Output

- **int puts(char *s)**
  - Print a string that s points out, Print `'\n' at the end`
  - Return a non-negative value if successful, otherwise return EOF
    - ✓ EOF (End Of File): Indicate the end of a file, holding -1

```
char str[10] = "Hi World";
int ret=1;

ret = puts(str);
printf("return: %d\n", ret);
```

Result

Print new-line '\n'

```
Hi World
return: 0
```

  - Use printf to print out str

5) Strings and Character Input/Output

- **char *gets(char *s)**
  - Read a string from a user, store the string in memory space that s points out, Return the pointer s
    - ✓ Store every character until receiving new-line ('\n')
    - ✓ Ignore the new-line '\n', put '\0' at the end
    - ✓ Should have enough space

```
char str[10];

gets(str);
printf("str: %s..", str );
```

Result

```
Hi World     ← Input
str: Hi World..
```

  - Use scanf to do the same

5) Strings and Character Input/Output

- (Note) gets is not available in the standard library
  - ✓ Visual Studio 2015 or later does not support
    - ✓ Use gets_s (Only in VS, non-standard function)
  - ✓ Use fgets function
    - ✓ Will cover later

5) Strings and Character Input/Output

- **int putchar(int c)**
  - Print out c
  - Return the character if successful, otherwise EOF
    - ✓ EOF (End Of File): -1

```
int ret = 1;

ret = putchar('a');
printf("\nreturn: %d\n",ret);

ret = putchar(99);
printf("\nreturn: %d\n",ret);
```

Result

```
a
return: 97
c
return: 99
```

5) Strings and Character Input/Output

- **int getchar(void)**
  - Read a character from a user

```
int c;

c = getchar();
putchar(c);
```

Result

```
H            ← Input
H
```