6
Character Strings

Why Learning Character Strings?

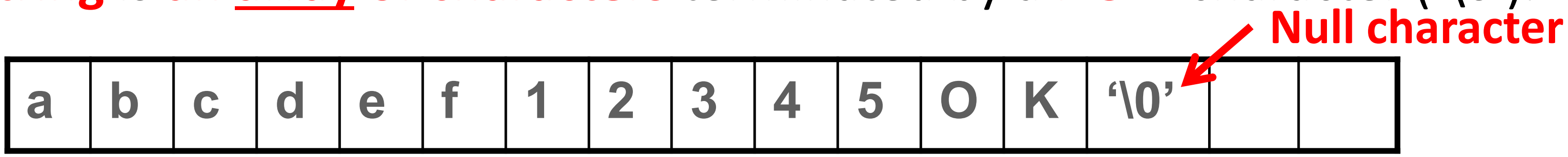
- In addition to handling numerical data, programs are also required to deal with alphabetical data.
- Strings are arrays of characters.
- C libraries provide a number of functions for performing operations on strings.
- In this lecture, string constants and string variables are first introduced. The different commonly used string functions from C libraries are then discussed.

Character Strings

- **String Declaration, Initialization and Operations**
- String Input and Output
- String Functions
- The ctype.h Character Functions
- String to Number Conversions
- Formatted String I/O
- Arrays of Character Strings

String Constants

- A **string** is **an array of characters** terminated by a **NULL** character ('\0').



Null character

a	b	c	d	e	f	1	2	3	4	5	O	K	'\0'		

- **String constant** is a set of characters in double quotes: e.g. **"C Programming"** - is an array of characters and automatically terminated with the **null** character **'\0'**
- Using #define to define a string constant: e.g. **#define** NTU **"Nanyang Technological University"**
- String constants can be used in function arguments, e.g. printf() and puts(): e.g. **printf("Hello, how are you?");**

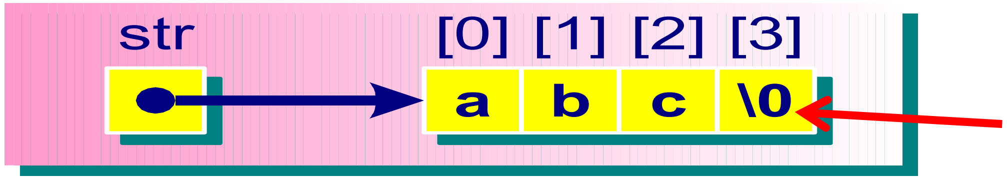
Note: Character Constant '**X**' vs String Constant "**X**":

– The character constant '**X**' consists of a single character of type **char**, while the character string constant "**X**" is an array of **char** that consists of two characters (i.e. the character '**X**' and the null character '**\0**').

String Variables: Declaration using Array Notation

- **String variables**: can be declared using **array notation**

(1) char str[ ] = "some text"; // ok
(2) char str[10] = "yes"; // ok
(3) **char str[4] = "four";** // **incorrect** -> null character missing
(4) **char str[ ] = {'a','b','c','\0'};** // ok, i.e. **char str[ ] = "abc";**



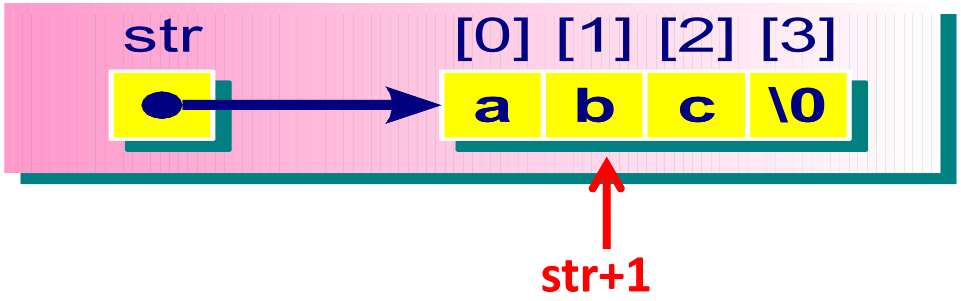
Null character

Note: **'\0'** differentiates a character string from an array of characters.

String Variables: Declaration using Array Notation

- Just like other kinds of arrays, the array name **str** gives the **address** of the 1st element of the array:

char str[ ] = "abc";



(1) str = = &str[0]

(2) ***str** = = 'a'

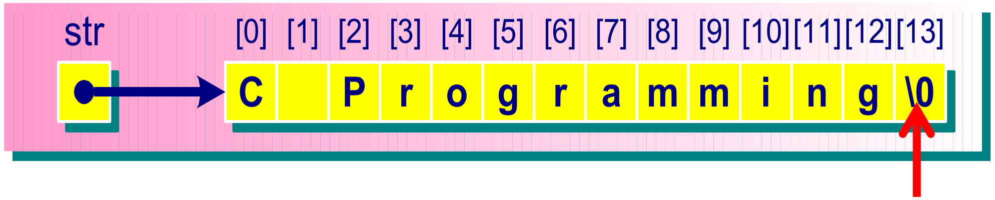
(3) ***(str+1)** = = str[1] = = 'b'

String Variables: Declaration using Pointer Notation

- **String variable** can also be declared using the pointer notation.
- When declaring a string variable using the pointer notation, we can assign a string constant to a pointer that points to the data type char:

  e.g. char *str = "C Programming";

- When a **string constant** is assigned to a pointer variable, C compiler will:
  1. Allocate **memory space** to hold the string constant.
  2. Store the **starting address** of the string in the pointer variable.
  3. Terminate the string with null ('\0') character.



Null character

String Variables: Array vs Pointer Declaration

- As can be seen earlier, there are two ways to declare a string:

(1) char **str1[ ]** = "How are you?"; //with **array** notation

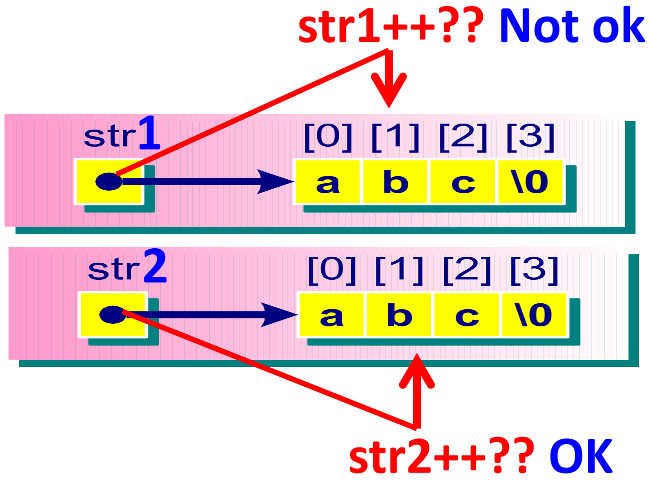
(2) char ***str2** = "How are you?"; //with **pointer** notation

Q: What is the difference between the two declarations?

str1: **pointer constant**, **str2**: **pointer variable**.

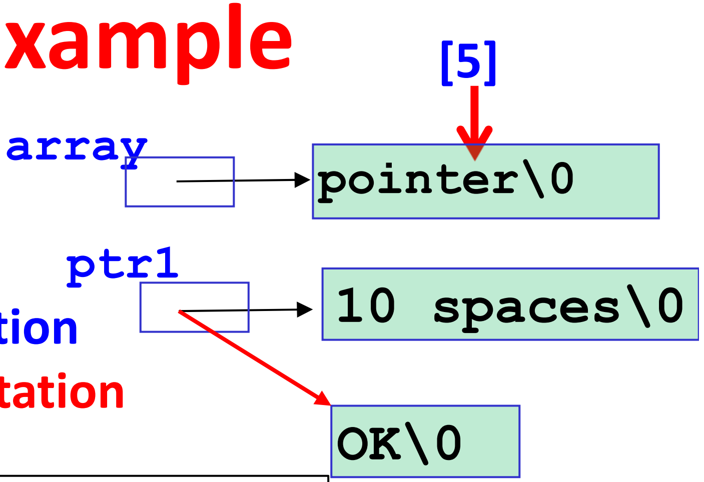
Therefore,

```
++str1;          // not OK
++str2;          // OK
str1 = str2;     // not OK
str2 = str1;     // OK
```



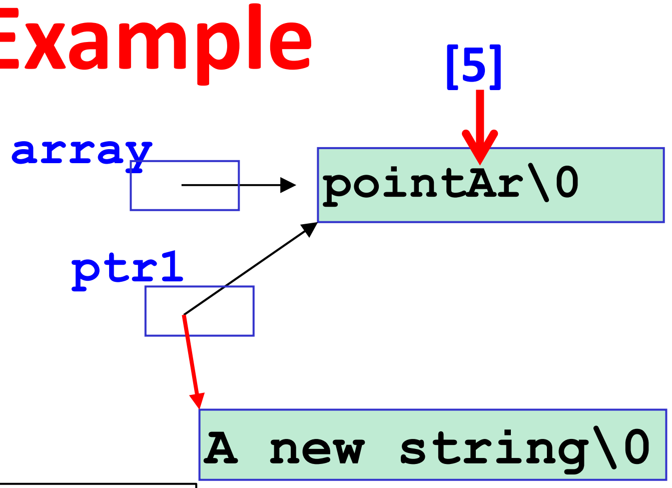
String Operations: Example

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";     // using array notation
    char *ptr1 = "10 spaces";     // using pointer notation

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);

    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);




    return 0;
}
```



```
ptr1 = 10 spaces
array = pointer
array = pointAr

ptr1 = OK
```

String Operations: Example



```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";    //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;
    printf("ptr1 = %s\n", ptr1);
    ptr1[5] = 'C';
    printf("ptr1 = %s\n", ptr1);
    ptr1 = "A new string";
    printf("ptr1 = %s\n", ptr1);
    return 0;
}
```

```
ptr1 = 10 spaces
array = pointer
array = pointAr

ptr1 = OK

ptr1 = pointAr
ptr1 = pointCr

ptr1 = A new string
```

Character Strings

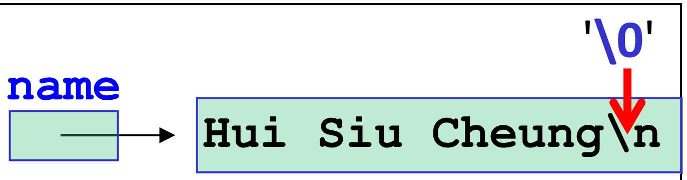
- String Declaration, Initialization and Operations
- **String Input and Output**
- String Functions
- The ctype.h Character Functions
- String to Number Conversions
- Formatted String I/O
- Arrays of Character Strings

String Input/Output

- There are 4 C library functions that can be used for string input/output:
  - **fgets()** (instead of gets()): function prototype **char *fgets(char *ptr, int n, FILE *stream);**
  - **puts()**: function prototype **int puts(const char *ptr);**
  - **scanf()**: function prototype **int scanf(control-string, argument-list);**
  - **printf()**: function prototype **int printf(control-string, argument-list);**
- The two most commonly used standard library functions for reading strings are **fgets()** and **scanf()**. For printing strings, the two standard library functions are **puts()** and **printf()**.
- Note that we use **fgets()** instead of **gets()** because **gets()** is not safe as it does not check the array bound.

String Input: fgets()

- **fgets() returns Null** if it fails, otherwise a pointer to the string is returned.
- Make sure **enough memory space** is allocated to hold the input string.



```
#include <stdio.h>
#include <string.h>
int main()
{
    char name[80], *p;     // allocate memory
    /*read name*/
    printf("Hi, what is your name?\n");
    fgets(name, 80, stdin);
    if ( p=strchr(name,'\n') )
       *p = '\0';      // replace ‘\n’ character in name
    /*display name*/
    printf(“Nice name, %s.\n", name);
    return 0;
}
```

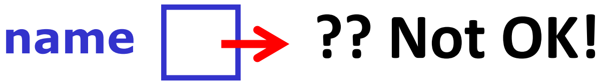
Output:

```
Hi, what is your name?
Hui Siu Cheung<\n>
Nice name, Hui Siu Cheung.
```

‘\0’

if: char *name;

Ok or not? Why?



String Output: puts()

```
#include <stdio.h>
#include <string.h>
int main( )
{
    char str[80], *p;           // string with allocated memory
    printf("Enter a line of string: ");
    if (fgets(str, 80, stdin) == NULL) {
            printf("Error\n");
    }
    if ( p=strchr(str,'\n') )  *p = '\0';
    puts(str);
    return 0;
}
```

***Input*:** **0123456789 OK**

0	1	2	3	4	5	6	7	8	9		O	K	‘\n’		

***Output*:** **0123456789 OK**

‘\0’

String Input/Output: scanf() and printf()

- **scanf()**
  - It reads the string up to the next **whitespace** character.
  - scanf() **returns** the **number of items** read by scanf(), otherwise **EOF** if fails.
  - Make sure that enough memory space is allocated for the input string.

- **printf()**
  - It **returns** the **number of characters** transmitted, otherwise a negative value will be returned if it fails.
  - It differs from the **puts()** function in that **no newline** is added at the end of the string.
  - The **printf()** function is less convenient to use than the **puts()** function. However, the **printf()** function provides the **flexibility** to the user to control the format of the data to be printed.

scanf() and printf(): Example

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;
    printf("Please enter your strings.\n");
    count = scanf("%s %s %s", name1, name2, name3);
    printf("I read the %d strings: %s %s %s\n", count, name1,
          name2, name3);
    return 0;
}
```

Output

```
Please enter your strings.
Hui Siu Cheung
I read the 3 strings: Hui Siu Cheung
```

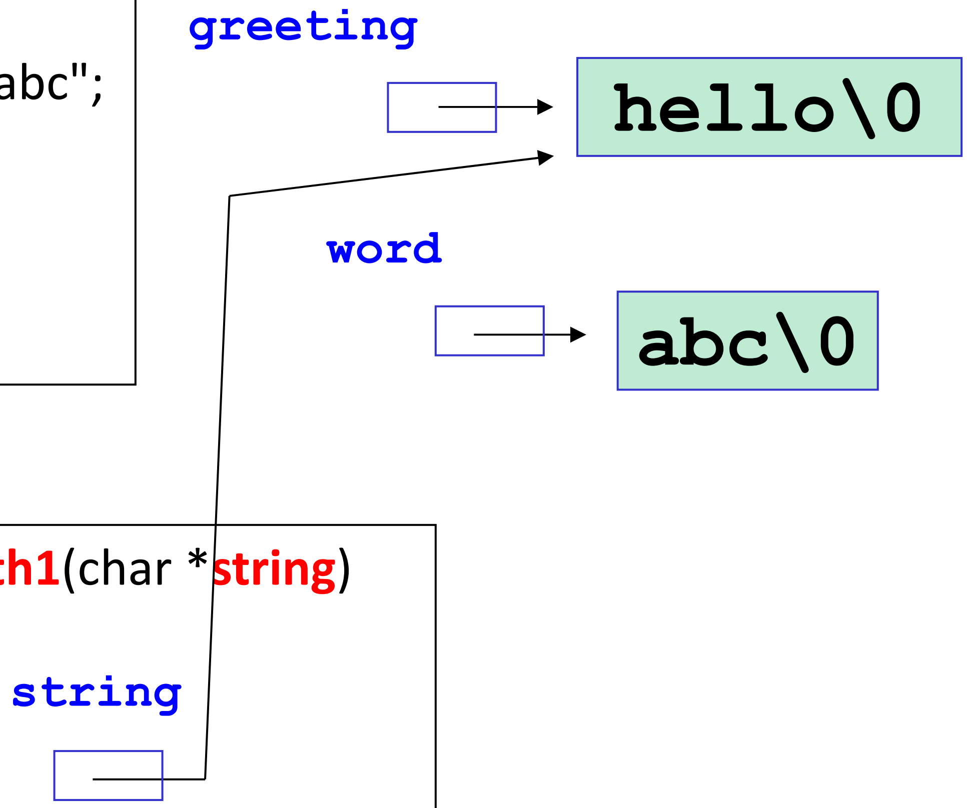
Separated by space

String Processing – Using Indexes

```
#include <stdio.h>
int length1(char []);
int main( )
{
      char   *greeting = "hello", word[] = "abc";
      printf(“The length is %d\n",
                    length1(greeting),
      return 0;
}
```

Output

```
The length is 5
```



using index notation

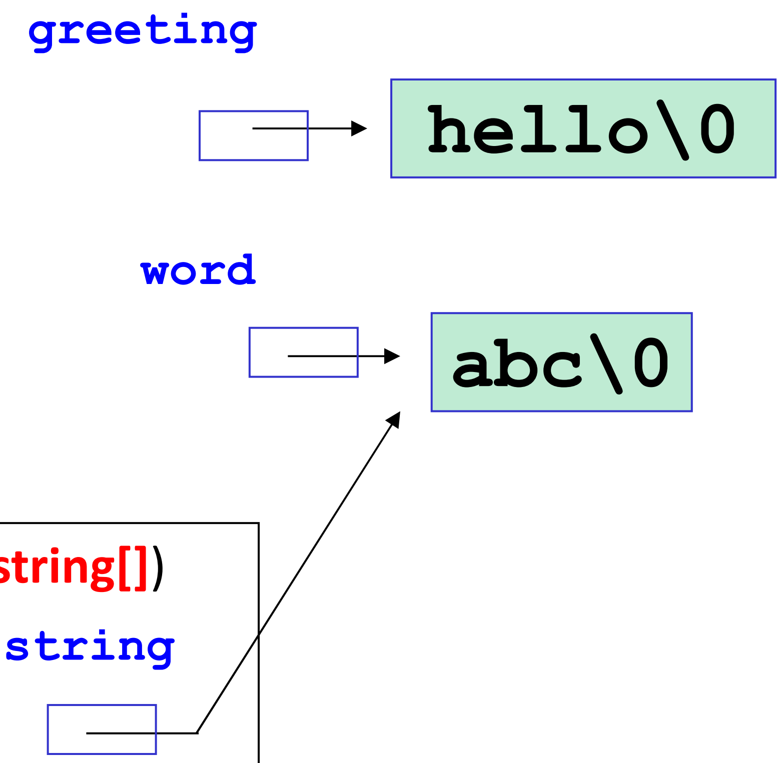
```
 int length1(char string[])  // or int length1(char *string)
{
      int count = 0;
      while (string[count] != ‘\0’)
           count++;
      return(count);
}
```

String Processing – Using Pointers

```
#include <stdio.h>
int length2(char *);
int main( )
{
     char   *greeting = "hello", word[] = "abc";
     printf(“The length is %d\n",
                length2(word));
     return 0;
}
```

Output

```
The length is 3
```



using pointer notation

```
int length2(char *string)   // or int length2(char string[])
{
       int count = 0;
       while ( *(string+count) != ‘\0’)
            count++;
       return(count);
}
```

Character Strings

- String Declaration, Initialization and Operations
- String Input and Output
- **String Functions**
- The ctype.h Character Functions
- String to Number Conversions
- Formatted String I/O
- Arrays of Character Strings

String Functions

- Must include the header file: **#include <string.h>**
- Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The strlen() Function

- The function prototype of **strlen** is

  size_t **strlen**(**const** char **str*);

  strlen computes and **returns** **the length of the string** pointed to by *str*, i.e. the number of characters that precede the terminating null character.

- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char line[81] = "This is a string";
    printf("The length of the string is %d.\n", strlen(line));
   return 0;
}
```

Output

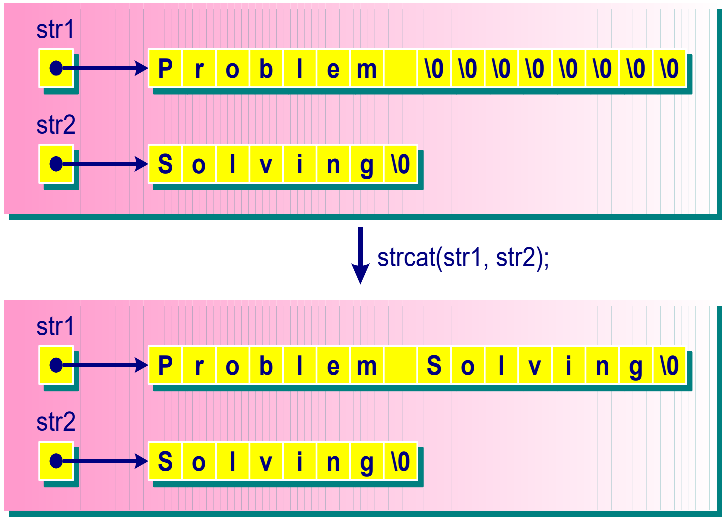
```
The length of the string is 16.
```

The strcat() Function

- The function prototype of **strcat** is

  char *strcat(char ***str1**, **const** char *str2);

  strcat **appends** a copy of the string pointed to by str2 to the end of the string pointed to by str1. The initial character of str2 overwrites the null character at the end of str1. strcat **returns** the value of **str1** (i.e. string).
- **Example**



str1
P r o b l e m \0 \0 \0 \0 \0 \0 \0 \0
str2
S o l v i n g \0
strcat(str1, str2);
str1
P r o b l e m S o l v i n g \0
str2
S o l v i n g \0

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[40] = "Problem ";
    char *str2 = "Solving";
    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```

Output

```
The first string: Problem
The second string: Solving
The combined string: Problem Solving
```

The strcpy() Function

- The function prototype of **strcpy** is

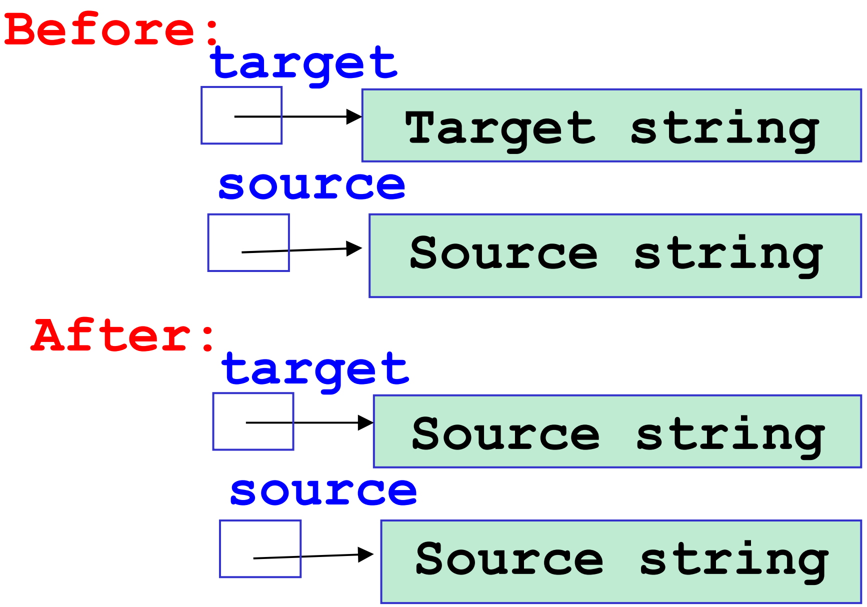
  ```
  char *strcpy(char *str1, const char *str2);
  ```

  strcpy copies the string pointed to by str2 into the array pointed to by str1. It **returns** the value of **str1** (i.e. string).

- Example

```
#include <stdio.h>
#include <string.h>
int main(){
    char target[40] = "Target string";
    char *source = "Source string.";
    puts(target); puts(source);
    strcpy(target, source);
    puts(target); puts(source);
    return 0;
}
```

```
Output
Target string
Source string
Source string
Source string
```



Before:

target

Target string

source

Source string

After:

target

Source string

source

Source string

The strcmp() Function

- The function prototype of **strcmp** is

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

strcmp compares the string pointed to by str1 to the string pointed to by str2.

- It **returns** an integer >, =, or < zero, accordingly if the string pointed to by str1 is >, =, or < the string pointed to by str2:
  - **0:** if the two strings are equal
  - **> 0 (the value could be the difference or 1 depending on system)**: if the first string follows the second string alphabetically, i.e. first string is larger (based on **ASCII values**)
  - **< 0 (the value could be the difference or -1 depending on system)**: if the first string comes first alphabetically, i.e. the first string is smaller (based on **ASCII values**)

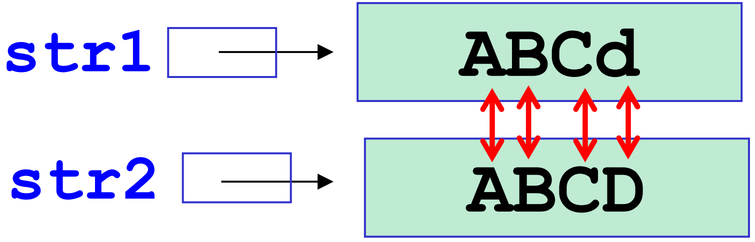
strcmp(): ASCII Character Set (Table)

	0	1	2	3	4	5	6	7	8	9
0	NUL							BEL	BS	TAB
1	LF		FF	CR						
2								ESC		
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

The strcmp() Function: Example 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81], *p;
    int result;
    printf("String Comparison:\n");
    printf("Enter the first string: ");
    fgets(str1, 81, stdin);
    if ( p=strchr(str1,'\n') ) *p = '\0';
    printf("Enter the second string: ");
    fgets(str2, 81, stdin);
    if ( p=strchr(str2,'\n') ) *p = '\0';
    result = strcmp(str1, str2);
    printf("The result of the comparison is
              %d\n\n", result);
    return 0;
}
```

Compare char by char using ASCII value in the strings:



str1

ABCd

str2

ABCD

Output

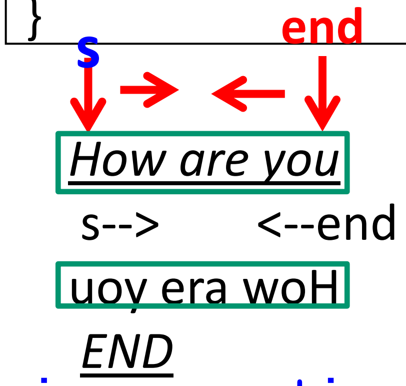
```
String Comparison:
Enter the first string: ABCd
Enter the second string: ABCD
The result of the comparison is 1

String Comparison:
Enter the first string: A
Enter the second string: AF
The result of the comparison is -1
```

Here, in this example, only 1, 0 or -1 is returned, it could also be the difference in ASCII values depending on the system.

The strcmp() Function: Example 2

```
/* Read a few lines from standard input &
write each line to standard output with
the characters reversed. The input
terminates with the line “END"*/
#include <stdio.h>
#include <string.h>
void reverse(char *);
int main()
{
    char line[132], *p;
    fgets(line, 132, stdin);
    if ( p=strchr(line,'\n') ) *p = '\0';
    while (strcmp(line, "END") !=0) {
            reverse(line);
            printf("%s\n", line);
            fgets(line, 132, stdin);
            if ( p=strchr(line,'\n') ) *p = '\0';
    }
}
```

```
void reverse(char *s)
{
    char c, *end;
    end = s + strlen(s) - 1;
    while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
          c = *s;
          *s++ = *end;    /*postfix op*/
                // i.e. *s = *end; s++;
          *end-- = c;
                // i.e. *end = c; end--;
    }
}
```



How are you

s-->   <--end

uoy era woH

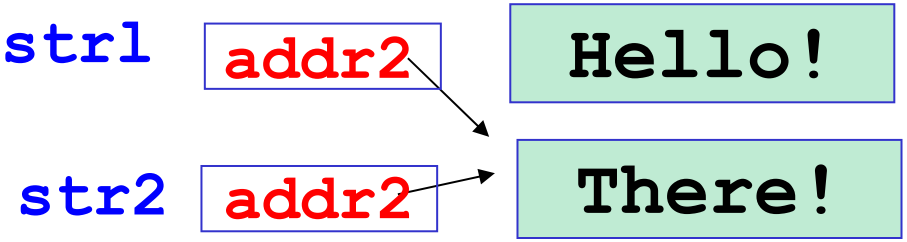
END

Swapping operation

Common Errors in Manipulating Strings

- When **copying** strings,

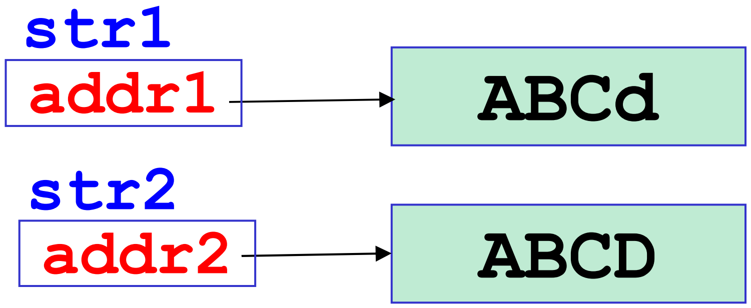
```
str1 = str2;
```



is **incorrect**, we should use: **strcpy(str1, str2);**

- When **comparing** two strings,

```
if (str1 == str2)  .......
```



is **incorrect**, we should use

if (strcmp(str1,str2) == 0) ...

Character Strings

- String Declaration, Initialization and Operations
- String Input and Output
- String Functions
- **The ctype.h Character Functions**
- String to Number Conversions
- Formatted String I/O
- Arrays of Character Strings

ctype.h Functions

Name	***True If Argument is***
Isalnum	Alphanumeric (alphabetic or numeric)
isalpha	Alphabetic
iscntrl	A control character, e.g. Control-B
isdigit	A digit
isgraph	Any printing character other than a space
islower	A lowercase character
isprint	A printing character
ispunct	A punctuation character (any printing character other than a space or an alphanumeric character)
isspace	A whitespace character: space, newline, formfeed, carriage return, etc.
isupper	An uppercase character
Isxdigt	A hexadecimal-digit character

- These functions are used to test the nature of a character.
- Return **true** **(non-zero)** if the character belongs to a particular class, and return **false** **(zero)** otherwise.
- Must include the header file: **#include <ctype.h>**

ctype.h: Character Conversion Functions

- **toupper()** - converts lowercase character to uppercase;
  tolower() - converts uppercase character to lowercase;

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void modify(char* str);
int main()   {
     char str[80], *p;                // allocate memory
     printf("Enter a string of text: \n");
     fgets(str,80,stdin); if ( p=strchr(str,'\n') ) *p = '\0';
     modify(str); puts(str);
     return 0;
 }
void modify(char* str) {
     while (*str != '\0') {
              if (isupper(*str))
                  *str = tolower(*str);
              else if (islower(*str))
                  *str = toupper(*str);
              str++;
     }
}
```

Output

This is a test

↑↑ →

t H…

tHIS IS A TEST

Character Strings

- String Declaration, Initialization and Operations
- String Input and Output
- String Functions
- The ctype.h Character Functions
- **String to Number Conversions**
- Formatted String I/O
- Arrays of Character Strings

String to Number Conversions

- There are two ways to store a number. It can be stored as strings or in numeric form. Sometimes, it is convenient to read in the numerical data as a string and convert it into the numeric form. To do this, C provides the functions: **atoi()** and **atof()**.
- Must include the header file: **#include <stdlib.h>**

atof( )

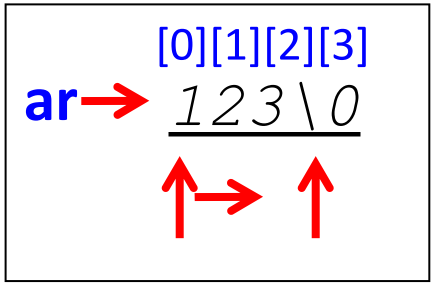
- Prototype: *double* ***atof*** *(**const** char *ptr)*;
- Functionality: converts the **string** pointed to by the pointer ***ptr*** into a double precision ***floating point number***.
- Return value: converted value.

atoi( )

- Prototype: *int* ***atoi*** *(**const** char *ptr)*;
- Functionality: *converts* the **string** pointed to by the pointer ***ptr*** *into an* ***integer****.*
- Return value: converted value.

String to Number Conversions: Example

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main()
{
   char  ar[80];
   int  i,   num;

   scanf("%s", ar);            //  read input string
   i=0;
   while (isdigit(ar[i])     // check digit in string
        i++;                 // until not a digit
   if (ar[i] != '\0')          // if not a null character
        printf("The input is not a number\n");
        /* for example, "1a2" */
   else {
        num = atoi(ar);
        printf("Input is %d\n", num);
   }
}
```



Output

```
123
Input is 123
```

Note:

- **atof()** and **atoi()** are useful when the program reads in a string and then converts the string into the corresponding number representation for further processing.
- Sometimes it is more convenient to read in a string instead of reading in a number directly.

Character Strings

- String Declaration, Initialization and Operations
- String Input and Output
- String Functions
- The ctype.h Character Functions
- String to Number Conversions
- **Formatted String I/O**
- Arrays of Character Strings

Formatted String I/O

The C standard I/O library provides two functions for performing formatted input and output **to strings**: **sscanf()** and **sprintf()**.

sscanf( )

- The function **sscanf()** is similar to scanf(). The only difference is that **sscanf()** takes input characters from a **string** instead of from the keyboard.
- The function **sscanf()** can be used to **transform numbers represented in strings**, e.g. the string "123" can be transformed into numbers 123 or 123.0 of data type int or double respectively.
- Function prototype: **sscanf(string_ptr, control-string, argument-list);**

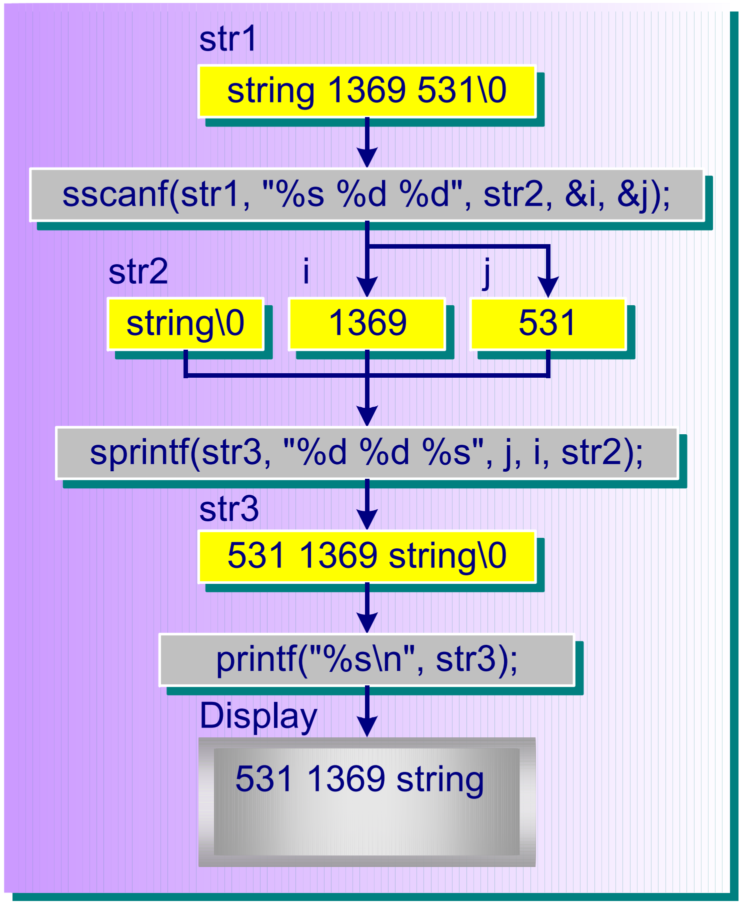
sprintf( )

- The function **sprintf()** is similar to printf(). The only difference is that **sprintf()** prints output to a **string**.
- **sprintf()** can be used to **transform numbers into strings**.
- Function prototype:

  sprintf(string_ptr, control-string, argument-list);

Formatted String I/O - Example

```
#include  <stdio.h>
#define   MAX_CHAR 80

int main()
{
     char  str1[MAX_CHAR] = "string 1369 531";
     char  str2[MAX_CHAR], str3[MAX_CHAR];
     int     i, j;

     sscanf(str1, "%s %d %d", str2, &i, &j);
     sprintf(str3, "%d %d %s", j, i, str2);
     printf("%s\n", str3);
     return 0;
}
```



str1

string 1369 531\0

sscanf(str1, "%s %d %d", str2, &i, &j);

str2 i j

string\0 1369 531

sprintf(str3, "%d %d %s", j, i, str2);

str3

531 1369 string\0

printf("%s\n", str3);

Display

531 1369 string

Output

```
531 1369 string
```

Character Strings

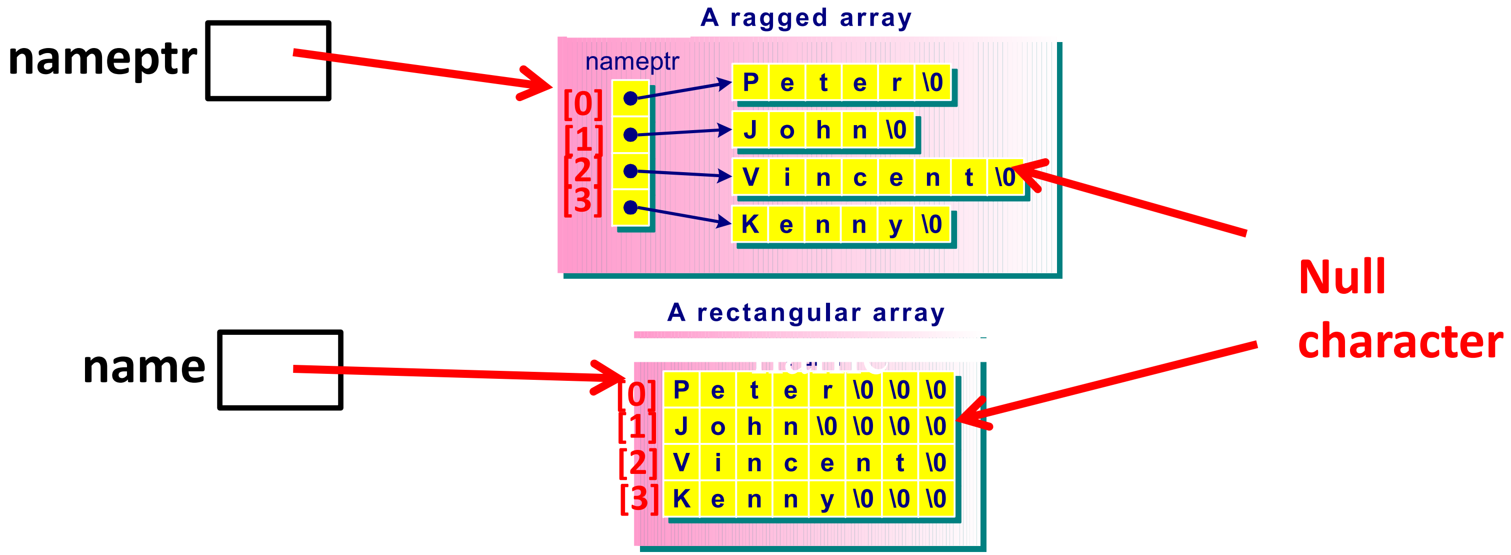
- – String Declaration, Initialization and Operations
- – String Input and Output
- – String Functions
- – The ctype.h Character Functions
- – String to Number Conversions
- – Formatted String I/O
- – **Arrays of Character Strings**

Array of Character Strings

- **Arrays of Character Strings [declared as array of <u>pointer variables</u>]**

```
char *nameptr[4] = {"Peter", "John", "Vincent", "Kenny"};
```

nameptr is a ***<u>ragged array</u>***, an *array of pointers* (save storage)



[0]	P	e	t	e	r	\0		
[1]	J	o	h	n	\0			
[2]	V	i	n	c	e	n	t	\0
[3]	K	e	n	n	y	\0		

[0]	P	e	t	e	r	\0	\0	\0
[1]	J	o	h	n	\0	\0	\0	\0
[2]	V	i	n	c	e	n	t	\0
[3]	K	e	n	n	y	\0	\0	\0

- **Arrays of Character Strings [declared as <u>2-D arrays</u>]**

```
char name[4][8]={"Peter","John","Vincent","Kenny"};
```

name is a ***<u>rectangular array.</u>***

Array of Character Strings: Example

```
#include <stdio.h>
int main()
{
    char *nameptr[4] = {"Peter", "John", "Vincent", "Kenny"};
    char name[4][10] = {"Peter", "John", "Vincent", "Kenny"};
    int  i, j;

    printf("Ragged Array: \n");
    for (i=0; i<4; i++)
      printf("nameptr[%d] = %s\n", i,
           nameptr[i]);

    printf("Rectangular Array: \n");
    for (j=0; j<4; j++)
      printf("name[%d] = %s\n", j,
          name[j]);
    return 0;
 }
```

Using for loop

Output

```
Ragged Array:
nameptr[0] = Peter
nameptr[1] = John
nameptr[2] = Vincent
nameptr[3] = Kenny
Rectangular Array:
name[0] = Peter
name[1] = John
name[2] = Vincent
name[3] = Kenny
```

Thank You!