# 11. Strings

[ECE10002/ITP10003] C Programming

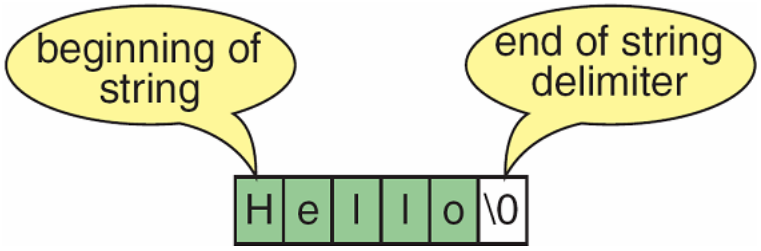
# Agenda

- Strings in C Language
- String Input/Output Functions
- String Manipulation Functions
- String/Data Conversion

# String Concepts

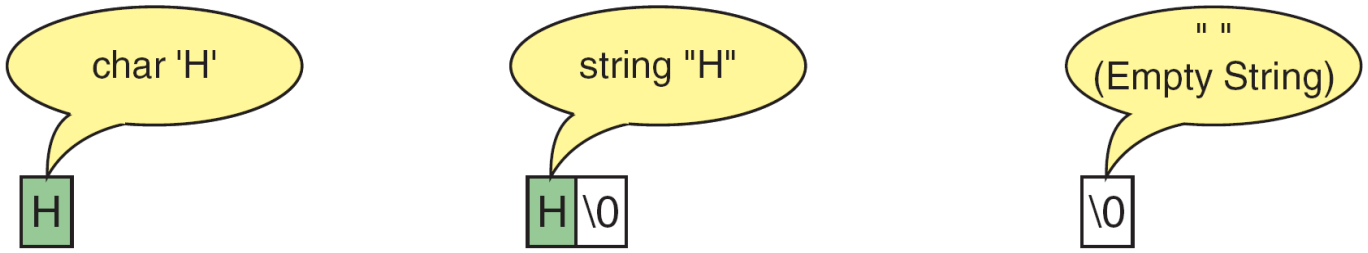
- String: ordered sequence of characters (or symbols)
  Ex) “Hello”, “Welcome to Handong Global University”

- String in C language
  - String literals are enclosed by double quotes.
  - String is represented by array of characters.
    Ex) char message[6] = “Hello”;
  - End of string is denoted by null character (‘₩0’)
    - Variable length string



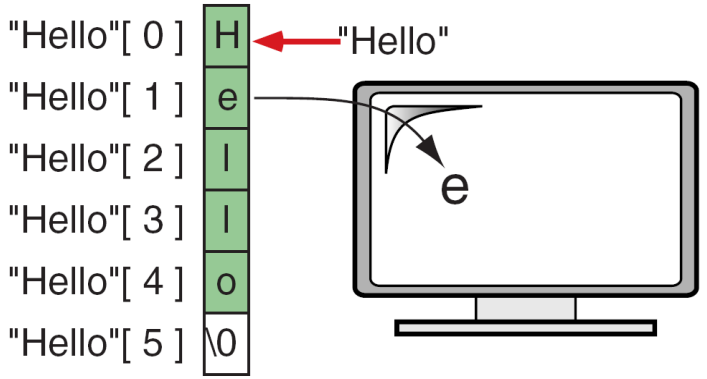
# Strings and Characters

- String vs. character




- String literals are referenced by the address of the first character

```
#include <stdio.h>
int main (void)
{
  printf("%c\n", "Hello"[1];
  return 0;
} // main
```

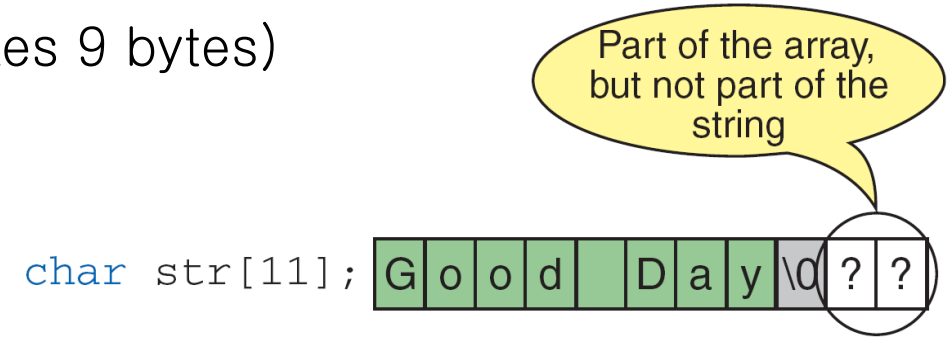


# Strings and Character Arrays

- String vs. character array
  - String terminates by null character '₩0'


- String length vs. array length

  Ex) char str[11] = "Good Day";
  - String length = 8 (takes 9 bytes)
  - Array length = 11



# Strings and Arrays

- A string is an array of characters that terminates with '₩0'.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| W | e | l | c | o | m | e | ! | ₩0 |

- For a string variable str,
  - str[i] is the $i^{th}$ character

    str[i] ≡ *(str + i)

  - &str[i] and str + i are the address of the $i^{th}$ character

    &str[i] == str + i // a substring

    // starting from the $i^{th}$ char.

# Example

- For a string str, str+i is a substring starting from the $i^{th}$ character of str.

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

int main()
{
    char str[256] = "Welcome!";
    int len = strlen(str);

    printf("str = %s₩n", str);

    for(int i = 0; i < len; i++){
        printf("str[%d] = %c, &str[%d] = %-16s, str + %d = %-16s₩n",
            i, str[i],
            i, &str[i],
            i, str + i);
    }

    return 0;
}
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| W | e | l | c | o | m | e | ! | ₩0 |

Result:

```
str = Welcome!
str[0] = W, &str[0] = Welcome!, str + 0 = Welcome!
str[1] = e, &str[1] = elcome! , str + 1 = elcome!
str[2] = l, &str[2] = lcome!  , str + 2 = lcome!
str[3] = c, &str[3] = come!   , str + 3 = come!
str[4] = o, &str[4] = ome!    , str + 4 = ome!
str[5] = m, &str[5] = me!     , str + 5 = me!
str[6] = e, &str[6] = e!      , str + 6 = e!
str[7] = !, &str[7] = !       , str + 7 = !
```

# Declaration and Initialization

- Declaration: same with declaration of character array

  Ex) char str[9];

  - Size should be <string length> + 1

- Initialization
  - char string[9] = “Good Day”;

    cf. char string[30] = “Good Day”;
  - char string[] = “Good Day”;
  - char *string = “Good Day”;
  - char string[] = { ‘G’, ‘o’, ‘o’, ‘d’, ‘ ’, ‘D’, ‘a’, ‘y’, ‘₩0’ };

Note! Assignment to a string is permitted only for initialization.

# String and Assignment Operator

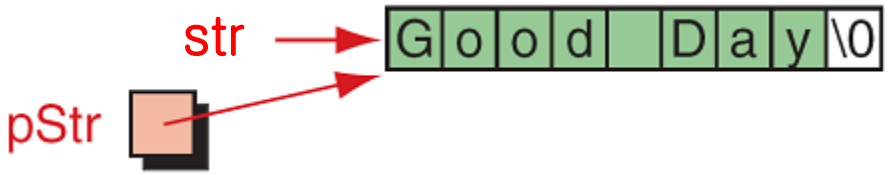
- Assigning to string constants

```
char str1[9] = "Good Day";   // Permitted only for initialization
char str2[9];
str2 = str1;                 // error!
str1 = "Good Day";           // error!
```

  - Array assignment is not allowed in C language!

- Assigning a string to a character pointer

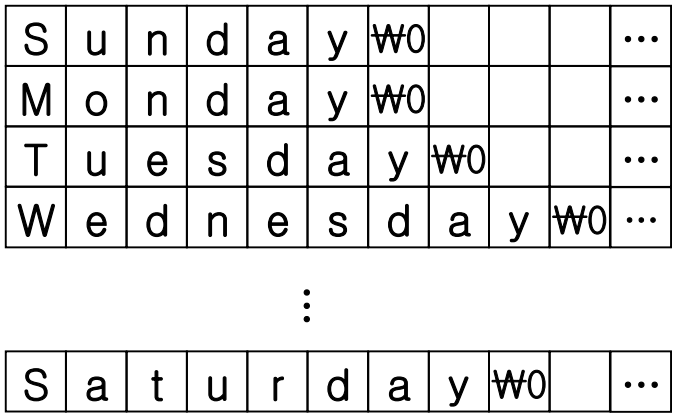
```
char str[9] = "Good Day";
char *pStr = NULL;
pStr = str;                  // OK
```



# Array of Strings

- Array of string: 2D array of char type

```
int i = 0;
char aDays[7][16] = {
   "Sunday",
   "Monday",
   "Tuesday",
   "Wednesday",
   ...
   "Saturday"
};
for(i = 0; i < 7; i++)
   printf("%s₩n", aDays[i]);
```



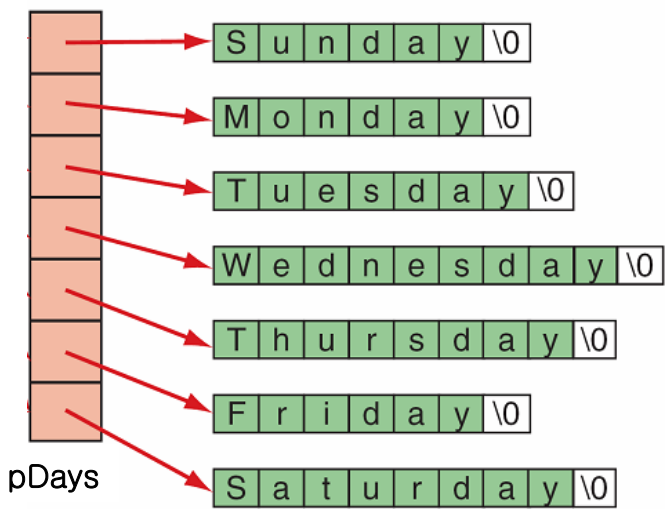
# Array of Strings

- Array of pointers

```
char *pDays[7];          // array of pointers
int i = 0;               // counter variable

pDays[0] = “Sunday”;
pDays[1] = “Monday”;
…
pDays[6] = “Saturday”;

for(i = 0; i < 7; i++)
   printf(“%s₩n”, pDays[i]);
```



# Agenda

- Strings in C Language
- String Input/Output Functions
- String Manipulation Functions
- String/Data Conversion

# String I/O Functions

- Formatted string I/O: printf, scanf

  Ex) char message[256];  // array size should be sufficient
  scanf("%s", message);  // & is not necessary
  printf("Message is %s", message);

  - Delimiter of scanf: white space characters
  - For safety, it is desirable to specify width modifier
    Ex) scanf("%255s", message);

# String I/O Functions

- Console string I/O
  - Input: char *gets(char* buffer);  // not safe
    - buffer: buffer to store the input string
      - Delimiter: '₩n', ('₩n' is replaced with '₩0')
    - return value
      - Success: buffer
      - Failure: NULL

    Note! Certain compilers prohibit gets() because it is unsafe.

  - Output: int puts(char* str);
    - strptr: string to print
      - '₩n' is appended automatically
    - Return value
      - Success: non-negative integer
      - Failure: EOF

# String I/O Functions

- String file I/O
  - Input: char *fgets(char *buffer, int buffer_size, FILE *sp);
    - buffer: buffer to store string
    - buffer_size: size of buffer
      - Maximum # of characters read: size – 1
    - sp: stream pointer
    - Return value
      - Success: strPtr
      - Failure: NULL
  - Output: int fputs(char *str, FILE *sp);
    - Return value
      - Success: 0
      - Failure: EOF

# Example

- Declaration

  char buffer[256];

  // input is [*Hello, World<Enter>*]

- Reading a word (“Hello,”)

  scanf(“%s”, buffer);

- Reading a text line (“Hello, World”)
  - Old method (unsafe)

  gets(buffer);

  - New method (recommended)

  fgets(buffer, 256, stdin);  // reads maximum 256 chars incl. ‘₩n’

  buffer[strlen(buffer)-1] = ‘₩0’;  // trim the last ‘₩n’

# Exercises

- Write a program that prints the content of a text file.

- data.txt

  Hello.

  Nice to see you.

  Welcome to Handong Global University.

- Result

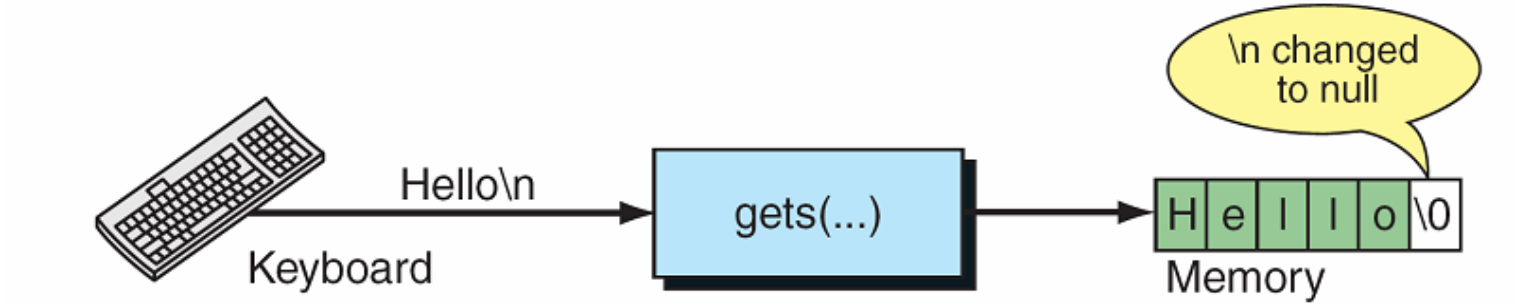
  0) Hello.

  1) Nice to see you.

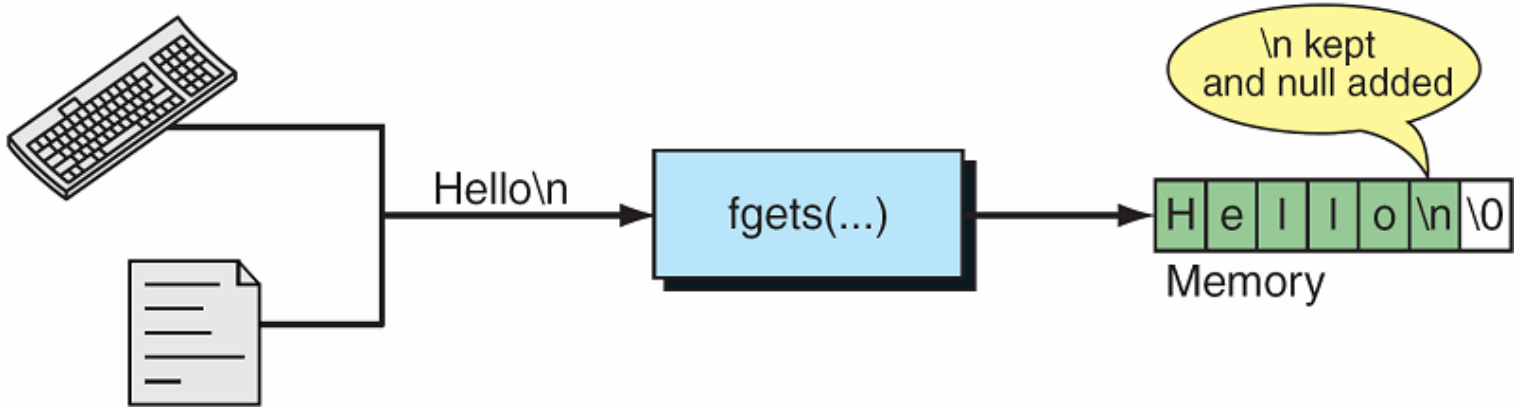
  2) Welcome to Handong Global University.

# gets() vs. fgets()

- gets() replaces '₩n' with '₩0'




- fgets() keeps '₩n' and appends '₩0'



# Agenda

- Strings in C Language
- String Input/Output Functions
- String Manipulation Functions
- String/Data Conversion

# String Manipulation Functions

- Given two strings str1, str2

```
char str1[10] = “123”;
char str2[10] = “456”;
char str3[10];
```

- Are these correct in C? NO!
  - Assignment or copy

```
str3 = str1;
```

  - Comparison

```
if(str1 == str2) …
if(str1 < str2) …
```

  - Concatenation

```
str3 = str1 + str2;          // Is the result “123456”?
```

# String Manipulation Functions

- String functions (declared in string.h)
  - String length: strlen
  - String copy: strcpy, strncpy, strcpy_s
  - String compare: strcmp, strncmp
  - String concatenation: strcat, strncat, strcat_s

# String Manipulation Functions

```
char str1[10] = "123";
char str2[10] = "456";
char str3[10];

// assignment of string, such as “str3 = str1;”
strcpy(str3, str1);

// comparison of strings
if(strcmp(str1, str2) == 0){  // if str1 and str2 have the same contents
    // statements
}

if(strcmp(str1, str2) < 0){    // if str1 precedes str2
    // statements
}

// concatenation of strings, such as “str3 = str1 + str2;”
strcpy(str3, str1);              // str3 == "123"
strcat(str3, str2);              // str3 == "123456"
return 0;
```

# Example

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

int main()
{
    char str1[128], str2[128], str3[128];

    printf("Input str2: ");
    fgets(str1, 128, stdin);            // reads a text line up to <Enter>
    str1[strlen(str1) - 1] = 0;
    printf("str1 = ₩"%s₩" (len = %d)₩n", str1, strlen(str1));

    printf("Input str2: ");
    fgets(str2, 128, stdin);            // reads a text line up to <Enter>
    str2[strlen(str2) - 1] = 0;
    printf("str2 = ₩"%s₩" (len = %d)₩n", str2, strlen(str2));

    if(strcmp(str1, str2) == 0)
        printf("str1 and str2 are the same.₩n");
    else
        printf("str1 and str2 are different.₩n");

    strcpy(str3, str1);
    printf("str3 = ₩"%s₩" (len = %d)₩n", str3, strlen(str3));

    strcat(str3, str2);
    printf("str3 = ₩"%s₩" (len = %d)₩n", str3, strlen(str3));

    return 0;
}
```

# String Length

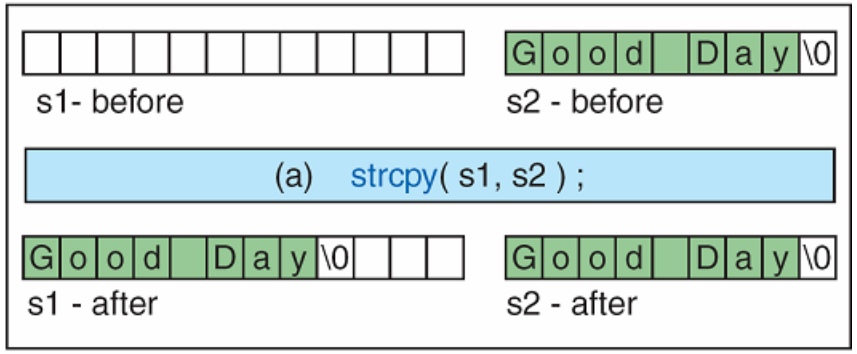
- Syntax: int strlen(const char *string);
  - string: input string
  - return value: length of string
    - '₩0' is not counted

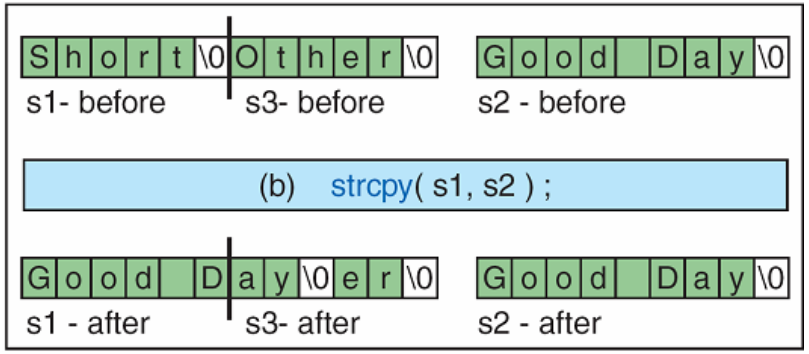
Ex)

```
char *string = "Hello";
printf("length of [%s] = %d₩n", string, strlen(string));
// result: length of [Hello] = 5
```

# String Copy

- Syntax
  - char* strcpy(char *toStr, const char* fromStr);
    - toStr: string buffer (destination)
    - fromStr: string to be copied (source)
    - Return value: toStr
  - Note! strcpy can be not safe!

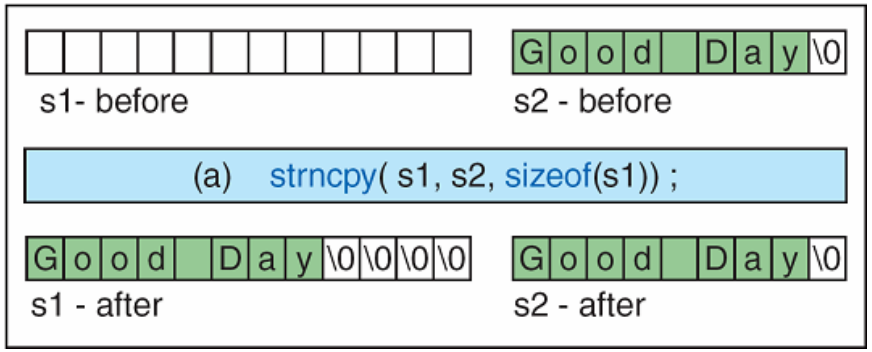


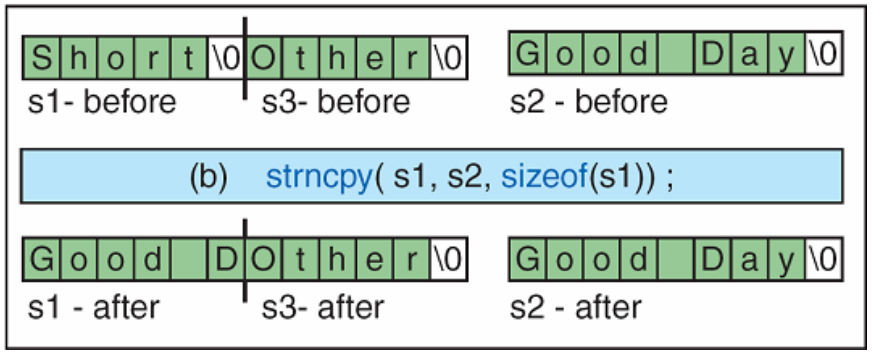




Copying long string

# String Copy

- String copy with length control
  - char* strncpy(char *toStr, const char* fromStr, size_t maxLen);
    - maxLen: maximum # of characters to copy
  - Note! If maxLen is not large enough, '₩0' can be omitted!



# String Compare

- String compare
  - 'less than' and 'greater than' relation of string are decided by alphabetical order

    Ex) "Hello" < "World", "abcde" > "ABCDE"

  - Syntax: int strcmp(const char*str1, const char*str2);
    - str1, str2: two strings to compare
    - Return value
      - 0: str1 and str2 stores the same string

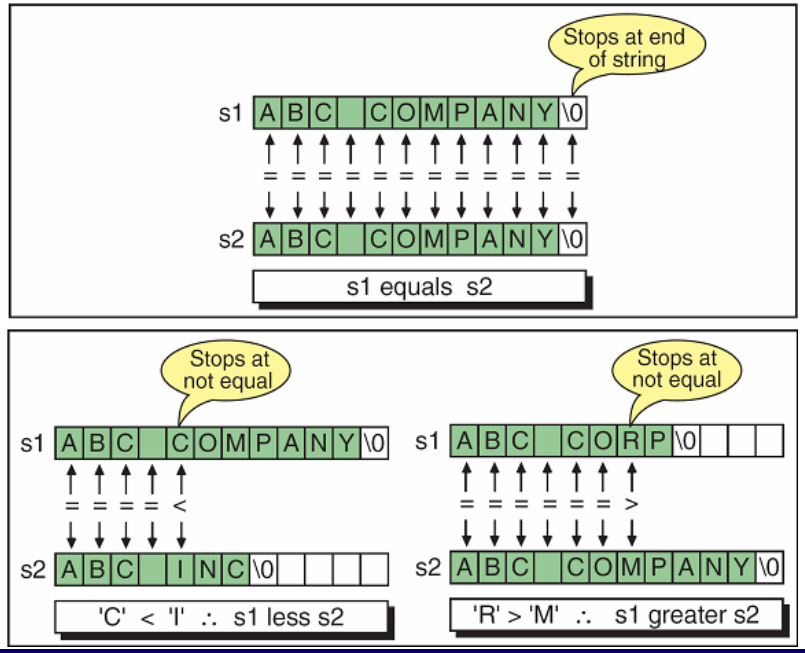
        strcmp("Hello", "Hello") == 0
      - Positive integer: str1 follows str2

        strcmp("abcde", "ABCDE") > 0
      - Negative integer: str1 precedes str2

        strcmp("Hello", "World") < 0

# String Compare

- Behavior of strcmp
  - Compares each character at str1 with the character at the same position in str2, from left to right.
    - If a difference is found, stop comparison, return the difference
    - If '₩n' is reached, return 0



# String Compare

- String compare with length limit
  - Syntax: int strncmp(const char *str1, const char *str2, int maxLen);
    - maxLen: maximum # of characters to compare

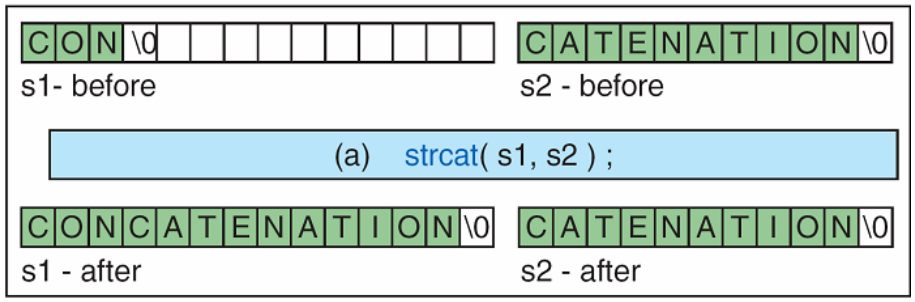
| string1 | string2 | Size | Results | Returns |
|---|---|---|---|---|
| "ABC123" | "ABC123" | 8 | equal | 0 |
| "ABC123" | "ABC456" | 3 | equal | 0 |
| "ABC123" | "ABC456" | 4 | string1 < string2 | < 0 |
| "ABC123" | "ABC" | 3 | equal | 0 |
| "ABC123" | "ABC" | 4 | string1 > string2 | > 0 |
| "ABC" | "ABC123" | 3 | equal | 0 |
| "ABC123" | "123ABC" | –1 | equal | 0 |

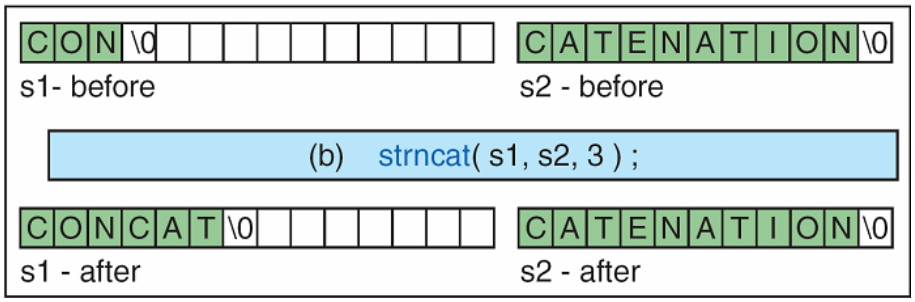
# String Concatenate

- String concatenation: appending a string to the end of another string

  Ex) “con” + “catenation” ➔ “concatenation”

  - Syntax: char* strcat(char* str1, const char* str2);
    - str2 is copied the end of str1
- Length-controlled
  - Syntax: char* strncat(char* str1, const char* str2, int maxLen);
    - maxLen: maximum # of characters to concatenate



# Agenda

- Strings in C Language
- String Input/Output Functions
- String Manipulation Functions
- String/Data Conversion

# String/Data Conversion

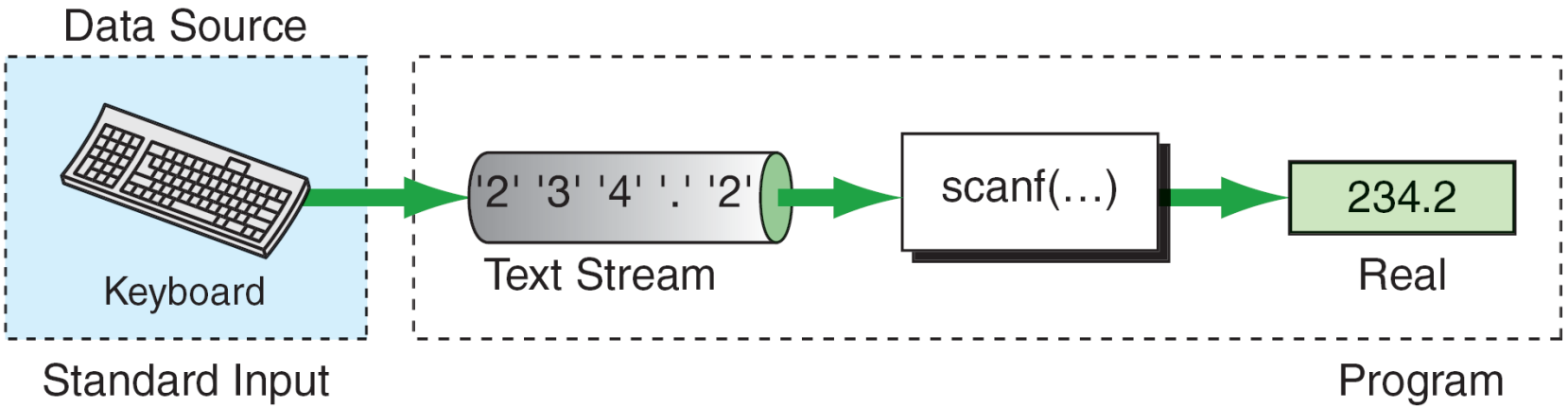
- Stream/Data conversion
  - Conversion from stream to values: fscanf, scanf
    - fscanf(fp, “%d”, &x);
  - Conversion from values to stream: fprintf, printf
    - fprintf(fp, “%d”, x);
- String/Data conversion
  - Conversion from string to values: sscanf
  - Conversion from values to string: sprintf
- String to value (integer, float, etc.)

  Ex) “256” ➔ 256, “3.14” ➔ 3.14F

# Formatted Input

- Formatted input: scanf
  - Inputs from the keyboard are sequences of characters
    - Value should be extracted from the text stream

    Ex) ‘2’, ‘3’, ‘4’, ‘.’, ‘2’ (character sequence) → 234.2 (float)
  - Function of scanf is the reverse of printf



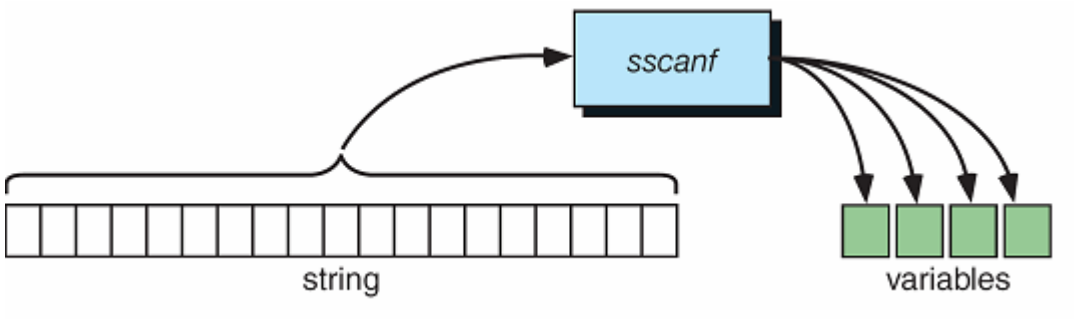
# String/Data Conversion

- Conversion from string to values
  - Syntax: int sscanf(char *str, const char* format_string, address_list);

  Ex) sscanf("35 x 50", "%d %*c %d", &width, &height);

  // %*c matches 'x' but discards it

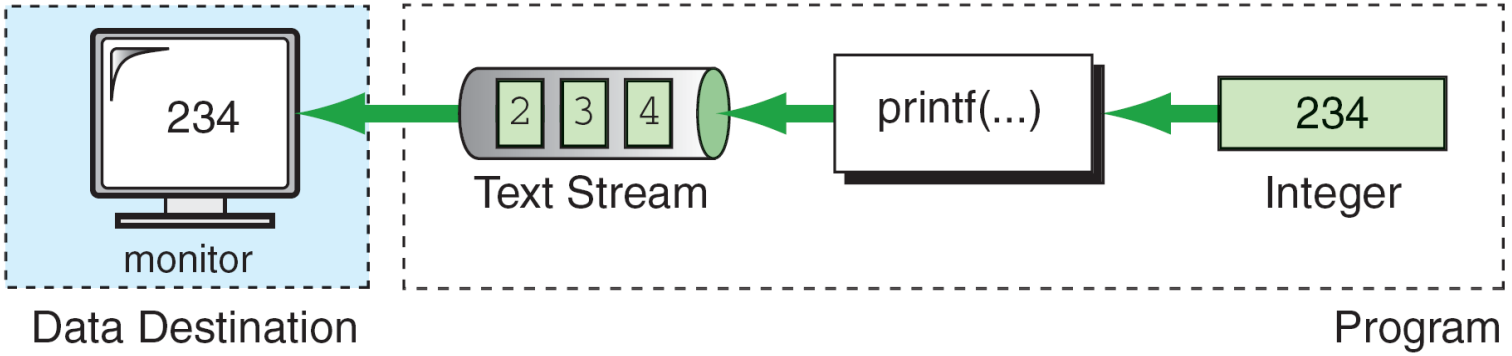
  - Header file: stdio.h



# Formatted Output

- Formatted output: printf
  - Monitor can display only text characters
    - Text data can be displayed directly, but numeral data requires formatting
  - Formatting: converting values to text stream

    ex) 234 (integer) → '2', '3', '4' (character sequence)

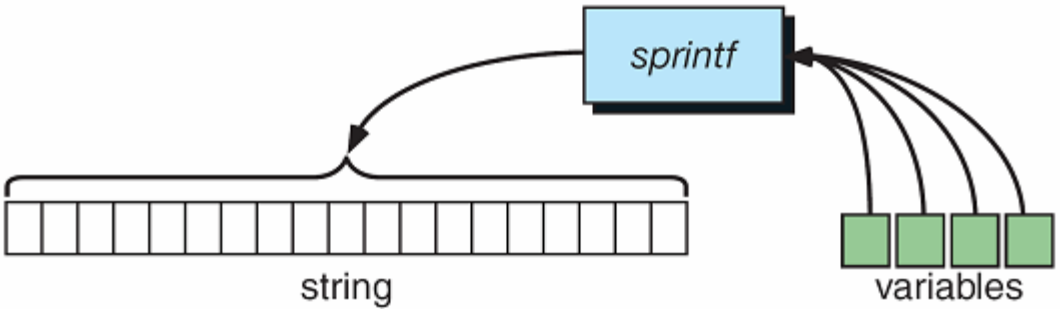


# String/Data Conversion

- Conversion from values to string
  - Syntax: int sprintf(char *str, const char* format_string, value_list);

Ex) char message[128];  // array size should be large enough
sprintf(message, “width = %d, height = %d₩n”, width, height);

  - Header file: stdio.h



# String to Integer/Float

- Header file: stdlib.h
- Conversion from string to integer
  - int atoi(const char *str);
    - Return value: converted value (If conversion fails, returns 0)

Ex)
```
char buffer[256];
int value = 0;
scanf("%255s", buffer);      // reads maximum 255 chars.
                             // buffer size should be 255 + 1
value = atoi(buffer);
```

- Conversion from string to long integer
  - long atol(const char *str);
- Conversion from string to float
  - float atof(const char *str);

# Example

- What does the following program do?

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

int main()
{
        int a = 0, b = 0, c = 0;
        char str1[30], str2[30], str3[60];

        printf("Input two numbers : ");
        scanf("%s %s", str1, str2);                    // assume the input is “123 456”

        a = atoi(str1);
        sscanf(str2, "%d", &b);

        c = a + b;

        strcpy(str3, str1);
        strcat(str3, str2);

        printf("%d + %d = %d₩n", a, b, c);
        printf("₩"%s₩" + ₩"%s₩" = ₩"%s₩"₩n", str1, str2, str3);

        return 0;
}
```