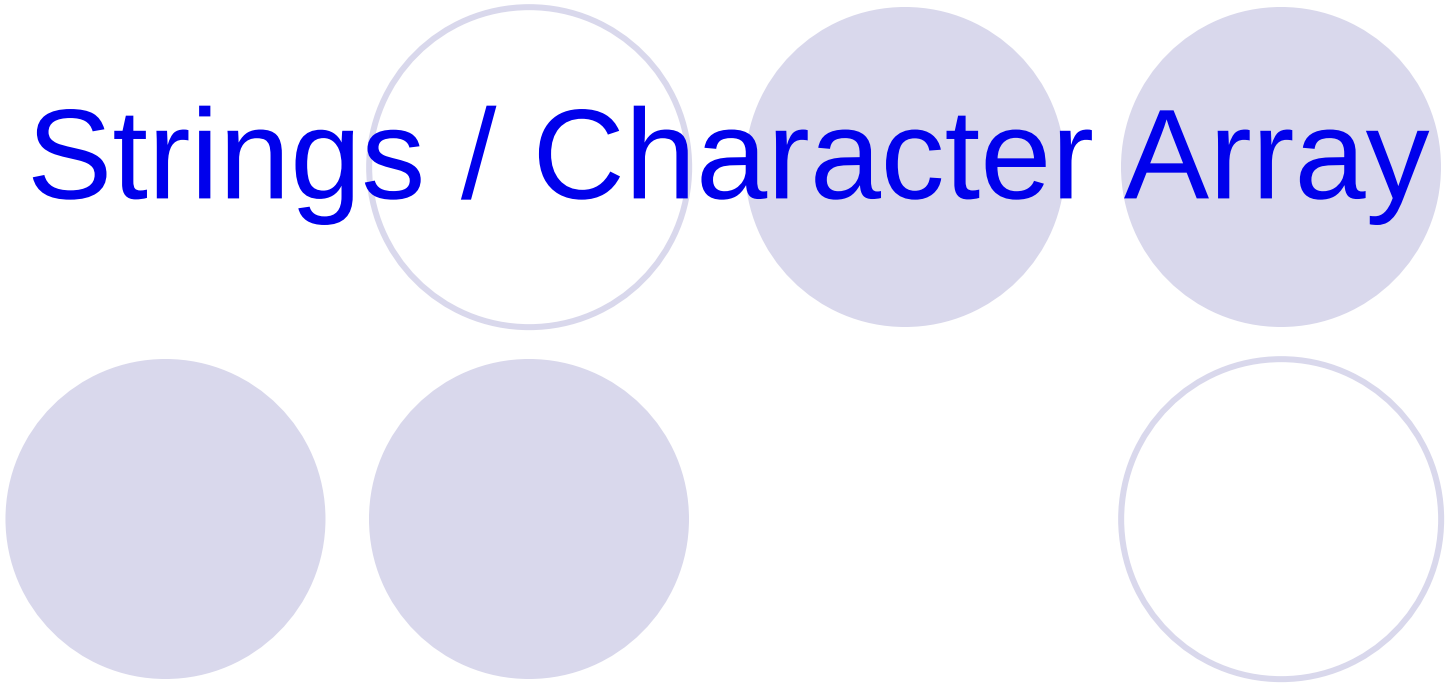
Strings / Character Array



Strings: one-dimensional character Arrays

- Strings in C are represented by arrays of characters.
- The end of the string is marked with a special character, the null character.
- The null of string-terminating character is represented by character escape sequence, \0;
- Although C does not have a string data type, it allows string constants. e.g. “hello world” is a string constant.

String declaration

- Strings can be declared like one-dimensional arrays.

e.g. **char str[30];**

char text[80];

Initialization of a string

- **String Initialization examples:**

(1) char str[9]="I like c";

which is the same as

char str[9]= {'I',' ','l','i','k','e',' ','c','\0' };

whenever a string, enclosed in double quotes, is written, C automatically creates an array of characters containing that string, terminating by the \0 character.

(2) char msg[]="Hello";

The size of the aggregate 'msg' is six bytes, five for the letters and one for the terminating NUL.

Initialization of a string

- **There is one special case where the null character is not automatically appended to the array. this is when the array size is explicitly specified and the number of initializers completely fills the array size.**

char c[4] = “abcd”;

Here, the array c hold only the four specified characters, a,b,c and d. No null character terminates the array.

Printing Strings

- Width and precision specifications may be used with the %s conversion specifier.
- the **width** specifies the minimum output field width; if the string is shorter, then space padding is generated.
- The **precision** specifies the maximum number of characters to display. If the string is too long, it is truncated.
- a *negative width* implies left justification of short strings rather than the default right justification.

e.g. printf("%7.3s", name);

- This specifies that only the first 3 characters have to be printed in a total field width of seven characters and right justified in the allocated width by default.

Example: Use of %s conversion specifier

```
main()
{
    char s[]="Hello, World";
    printf(">>%s<<\n", s);
    printf(">>%20s<<\n", s);
    printf(">>%-20s<<\n",s);
    printf(">>%.4s<<\n",s);        /* >>hell<<   */
    printf(">>%-20.4s<<\n",s);
    printf(">>%20.4s<<\n",s);
}
---Another way for printing string.----
main()
{
    char s[]="Hello World";
    puts(s);
}
```

String input

- Using %s control string with scanf()
- With scanf() for input the strings, two limitations:
  - scanf() only recognizes a sequences of characters delimited by white space characters as an external string.
  - [White space characters- space, formfeed, new-line, carriage return, horizontal tab or vertical tab]
  - it is the programmer's responsibility to ensure that there is enough space to receive and store the incoming string along with the terminating null which is automatically generated and stored by scanf().

String input: using scanset

- The scanset conversion facility provided by scanf() is a useful string input method.
- It allows the programmer to specify the set of characters that are (or are not) acceptable as part of the string.
- A scanset conversion consist of a list of acceptable characters enclosed within square brackets.
- A range of characters may be specified using notations such as ‘a-z’, meaning all characters within this range.

e.g.

```
main()
{
    char str[50];
    scanf(“%[a-z]”, str);
    printf(“The string was: %s\n”, str);
}
```

Single-line input using scanset with ^

- the circumflex (^) plays an important role while taking input.
- if the first character after the '[' is a '^' character, then the rest of the scanset specifies unacceptable characters rather than acceptable characters.

Example:

```
main()
{
    char str[80];
    scanf("%[^\n]", str);  /*It ends string with "\n" when user press
                                                enter key. */

    printf("The string was: %s\n", str);
}
```

Multiline input using scanset

- one can use a bracketed string read, %[..] where [ ] are used to enclose all characters which are permissible in the input.
- If any character other than those listed within the brackets occurs in the input string, further reading is terminated.
- Such input terminators must be preceded by the caret (^).
- e.g.

```
main()
{
    char string[80];
    scanf(“%[^~]”, string);
    printf(“%s”, string);
}
```

gets()

```
main()
{
	char str[150];
	gets(str);
	printf(“The string was %s\n”, str);
}
```

Character Handling Library

(include heaser file “ctype.h”)

	Function Prototype	Function description
1	int isdigit(int c)	Returns a true value if c is a digit and 0 (false) otherwise.
2	int isalpha(int c)	Returns a true value if c is a letter and 0 otherwise.
3	int isalnum(int c)	Returns a true value if c is a digit or letter, and 0 otherwise.
4	int isxdigit(int c)	Returns a true value if c is a hexadecimal digit character and 0
5	int islower(int c)	Returns a true value if c is a lowercase letter and 0 otherwise.
6	int isupper(int c)	Returns a true value if c is an uppercase letter and 0 otherwise.
7	int tolower(int c)	If c is an uppercase letter, tolower() returns c as a lowercase letter. Otherwise, tolower() returns the argument unchanged.
8	int isspace(int c)	Returns a true value if c is a white space character such as newline(‘\n’), space(‘ ‘), form feed(‘\f’), carriage return(‘\r’), horizontal tab(‘\t’) or vertical tab(‘\v’) and 0 otherwise.
9	int iscntrl(int c)	Returns a true value if c is a control character and 0 otherwise.
10	int ispunct(int c)	Returns a true value if c is printing character other than a space, a digit, or a letter and 0 otherwise.
11	int isprint(int c)	Returns a true value if c is a printing character including space (‘ ‘), and 0 otherwise.
12	int isgraph(int c)	Returns a true if c is a printing character other than space (‘ ‘), and 0 otherwise.

Table X.1: Summary of the character handling library function

String Manipulation Functions of The String Handling Library

Function prototype	Function description
strcpy(s1, s2)	Copies the string s2 into the array s1. The value of s1 is returned.
strncpy(s1,s2, n)	Copies at most n characters of the string s2 into the array s1. The value of s1 is returned.
strcat(s1, s2)	Appends the string s2 to the array s1. The first character of s2 overwrites the terminating NULL character of s1. The value of s1 is returned.
strncat(s1,s2, n)	Appends at most n characters of string s2 to array s1. The first character of s2 overwrites the terminating NULL character of s1. The value of s1 is returned.

Table X.4: The string manipulation functions of the string handling library

`strlen(s)`	Determines the length of string **`s`**. The number of characters preceding the terminating `NULL` character is returned.

Comparison Functions Of The String Handling Library

Function prototype	Function description
strcmp(s1, s2)	Compares the string s1 to the string s2. The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively.
strncmp(s1,s2, n)	Compares up to n characters of the string s1 to the string s2. The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively.

Table X.5: The string comparison functions of the string handling library

Exercises

- Write a ‘c’ program to input one text string (multi line input). Do the following for the string.

  - Count number of words

  - Number of lines

  - Count number of frequencies of each word.

  - Sort the words (h.w.)