Lecture 15:

Strings

Prof. Shervin Shirmohammadi
University of Ottawa

Character String

- A character string is a sequence of characters (essentially a sequence of `chars`)treated as a single unit.
- In C, a character string in a program can be delimited by a pair of double quotes " and "
  - E.g.:

```
"This is a character String."
printf("Welcome to the University of Ottawa.\n");
```

  - The last character in a string must be the Null Character `'\0'`.
    - Not to be confused with the NULL address
  - The name of the *array* used to store the string is in fact the name of the string.
  - Recall that the name of an array is also **an address** that points to the first element in the array; in this case, this is the first character in the string.
  - **In C, the expression of a string enclosed in quotes is represented by its address – thus the first argument of printf is evaluated to an address.**

String Declaration and Output

- A string may be declared and initialized in one of many ways.

  E.g.: `char color[] = "red";`

  `char color[] = {'r', 'e', 'd', '\0'};`

- Note that if we use a pair of " in the assignment of a string to an array of `char`, we do not need to include the null character explicitly as C will add it automatically for us.
- A string can be printed out using `printf` and the `%s` conversion specifier.

  E.g.: `printf("%s", color);`

  - `printf` will stop printing out characters when it hits the null character.
- **All C standard functions for manipulating strings use the null character as the delimiter**.
  - ➔ If the null character is missing, the program will continue processing the array of `char` going beyond the array limits until it hits a null character somewhere in memory! This could lead to strange effects.

String Input

- We can read in a string from the keyboard using `scanf` and the `%s` conversion specifier.

```
E.g.:   char text_in[10];
        scanf("%s", text_in);
```

  - Note that there is **no ampersand** `&` preceding the name `text_in` in the above.
    - Recall that an array name without the `[]` is an address and in this case the address refers to the memory area where the string is to be stored.
  - `text_in` may contain at most 10 characters, including the null character; thus the user can enter at most 9 characters.
  - `scanf` will read characters until the next space, carriage return, or `EOF`.
    - To read beyond spaces, one can use the `gets` function, with prototype `gets(char *)`.
  - `scanf` leaves the space and/or new line characters in the input stream.

- Individual elements of a string are of course accessed just like the elements of any array.

```
E.g.:   char hello[] = "Greetings";
        printf("%c\n", hello[0]);
```

  - A string can thus be printed out using a series of calls to the function `putchar`.

```
E.g.:   putchar(hello[ctr]);
```

- Careful: do not confuse the null character `'\0'` with the symbolic constant `NULL` which corresponds to the null address.

Exercise

- Write a program that prompts the user for a name and prints back a greeting message.

Manipulating Characters and Strings

- There exists in C a large number of standard functions for manipulating characters and strings.
- The header file `ctype.h` contains the prototypes of a number of functions useful for manipulating or testing **one character**.
  - See the next slide for a summary of these functions.
- The header file `string.h` contains the prototype of functions for manipulating **a string** of characters.
  - These functions allow, among many, to compare strings, to search strings, and to find the length of strings.

`int isdigit(int c)`	Returns a true value if `c` is a digit, and 0 (false) otherwise.
`int isalpha(int c)`	Returns a true value if `c` is a letter, and 0 otherwise.
`int isalnum(int c)`	Returns a true value if `c` is a digit or a letter, and 0 otherwise.
`int isxdigit(int c)`	Returns a true value if `c` is a hexadecimal digit character, and 0 otherwise. (See Appendix E, "Number Systems," for a detailed explanation of binary numbers, octal numbers, decimal numbers, and hexadecimal numbers.)
`int islower(int c)`	Returns a true value if `c` is a lowercase letter, and 0 otherwise.
`int isupper(int c)`	Returns a true value if `c` is an uppercase letter, and 0 otherwise.
`int tolower(int c)`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.
`int toupper(int c)`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace(int c)`	Returns a true value if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`), or vertical tab (`'\v'`)—and 0 otherwise
`int iscntrl(int c)`	Returns a true value if `c` is a control character, and 0 otherwise.
`int ispunct(int c)`	Returns a true value if `c` is a printing character other than a space, a digit, or a letter, and 0 otherwise.
`int isprint(int c)`	Returns a true value if `c` is a printing character including space (`' '`), and 0 otherwise.
`int isgraph(int c)`	Returns a true value if `c` is a printing character other than space (`' '`), and 0 otherwise.

- The header file `stdlib.h` contains the prototype of functions for converting a string of digits to an integer or a real number.
  - The figure below summarizes these functions.

`double atof(const char *nPtr)`	Converts the string `nPtr` to `double`.
`int atoi(const char *nPtr)`	Converts the string `nPtr` to `int`.
`long atol(const char *nPtr)`	Converts the string `nPtr` to `long int`.
`double strtod(const char *nPtr, char **endPtr)`	Converts the string `nPtr` to `double`.
`long strtol(const char *nPtr, char **endPtr, int base)`	Converts the string `nPtr` to `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base)`	Converts the string `nPtr` to `unsigned long`.

Example

- Write a program that prompts the user for a sentence and converts it to the standard title format (first character of each word in upper case and all other characters lower case).

String Manipulation Functions

- The header file `string.h` contains the prototype of functions for string manipulation.
  - The figure below summarizes these functions.

```
strlen(char *x);

strcpy(char *dst, char const *src);
strncpy(char *dst, char const *src, int N);

strcat (char *dst, char const *src);
strncat (char *dst, char const *src, int N);

int strcmp(char const *s1, char const *s2);
int strncmp(char const *s1, char const *s2, int N);
```