Week 4

XMUT-NWEN 241 - 2024 T2

Systems Programming

Mohammad Nekooei

School of Engineering and Computer Science

Victoria University of Wellington

String

Assigning a string after array declaration

```
char str[10];
…
str = "Hello !";
```

Illegal! Use `strcpy()` function

```
char str[10];
…
strcpy(str, "Hello !");
```

Strings

- `long int strlen(const char* source);`
  - Calculates the length of a given string, up to the first null character.
- `char* strcpy(char* destination, const char* source);`
  - Copies the source string to the destination character array.
- `int strcmp (const char* str1, const char* str2);`
  - Compares two strings and returns 0 if both strings are identical.
- `char *strcat(char *dest, const char *src);`
  - Concatenates two strings and stores the result in the first argument.

Null Terminator '\0'

- A string is an array of characters that ends with the first occurrence of `'\0'`
- What comes **after** the end of the string **doesn't matter**, since the string has ended

```
char str[] = "One\0Two";
printf("%s\n", str);
```

- The program will print only the string "One"
  - The `'\0'` character terminates the string
  - What comes after, does not matter
- The array will contain 8 elements

Displaying Strings: `printf()`

- Strings can be displayed on the screen using `printf()`

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

- The **precision** (`'%.N'`) parameter limits the length of longer strings to at most `N`

```
printf("%.5s\n", "abcdefg");
   // only "abcde" will be displayed
```

- The **width** (`'%N'`) parameter can be used to print a short string in a long space, at least `N` characters

```
printf("%5s\n", "abc" );
   // prints "  abc". Note the leading
   // two spaced at the beginning.
```

Displaying Strings: puts()

- The puts() function writes the string out to standard output and automatically appends a newline character at the end

```
char str[] = "This is an ";
printf("%s", str);
puts("example string.");
printf("See??\n");
```

- The output will be:

```
This is an example string.
See??
```

Reading in strings – scanf()

- The standard format specifier for reading strings with `scanf()` is `%s` that the `'&'` is **not required** in the case of strings, since the string is a memory address itself

- `scanf()` appends a '\0' to the end of the character string stored

- `scanf()` does skip over any leading whitespace characters in order to find the first non-whitespace character

Reading in strings – scanf()

- The width field can be used to limit the maximum number of characters to read from the input
- You should use one character less as input than the size of the array used for holding the result

```
char str[6];
printf("Hi\n");
scanf("%5s", str);
   // If you enter "HelloBello123xyz", only the
   // first 5 characters will be read and a
   // concluding '\0' will be put at the end

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

Reading in strings – scanf()

- `scanf()` reads in a string of characters, only up to the first non-whitespace character
  - it stops reading when it encounters a space, tab, or newline character
- C supports a format specification known as the edit set conversion code `%[...]`
  - it can be used to read a line containing a variety of characters, including white spaces

```
char str[20];
printf("Enter a string:\n");
scanf("%[^\n]", str);
printf("%s\n",str);
```

Reading in strings – scanf()

- Always use the width field to limit the maximum number of characters to read with `"%s"` and `"%[...]"` in all production quality code!
  - No exceptions!

Reading in strings – gets()

- `gets()` is used to scan a line of text from a standard input device, until a newline character input
- **The string may include white space characters**
- The newline character won't be included as part of the string
- `'\0'` is always appended to the end of the string of stored characters

Reading in strings – gets()

```
char str[15];
printf("Enter your name: \n");
gets(str);
printf("%s\n", str);
```

- `gets()` has no provision for limiting the number of characters to read
  - This can lead to overflow problems!

Reading strings character by character

- Read in character by character is useful when
  - you don't know how long the string might be,
  - or if you want to consider other stopping conditions besides spaces and newlines
    - e.g. stop on periods, or when two successive slashes, //, are encountered.

- The `scanf()` format specifier for reading individual characters is `%c`

- If a width greater than 1 is given (`%2c`), then multiple characters are read, and stored in successive positions in a char array

sscanf() and sprintf() functions

- `scanf()` and `printf()` functions are used to read from and write to the standard input/output

- `sscanf()` and `sprintf()` are used for the same goal but instead of the standard input/output, **they use strings**

- One of their main advantage is when you need to prepare a string for later use

The <ctype.h> header

- `<ctype.h>` declares a set of functions to classify and transform individual chars
  - `#include <ctype.h>` is required to use any of these functions
  - https://www.tutorialspoint.com/c_standard_library/ctype_h.htm documents the library

The <ctype.h> header

- Some of the more commonly used functions:
  - `isupper()` – checks if a character is an uppercase letter
    - A value different from zero is returned if the character is an uppercase alphabetic letter, zero otherwise
  - `islower()` – checks if a character is a lowercase letter
    - A value different from zero is returned if the character is a lowercase alphabetic letter, zero otherwise
  - `toupper()` – converts a character to its uppercase equivalent if the character is an lowercase letter and has an uppercase equivalent
    - If no such conversion is possible, the returned value is unchanged
  - `tolower()` – converts a character to its lowercase equivalent if the character is an uppercase letter and has a lowercase equivalent
    - If no such conversion is possible, the returned value is unchanged

The <string.h> header

- `<string.h>` defines several functions to manipulate null-byte terminated arrays of chars
  - `#include <string.h>` is required to use any of these functions
  - https://www.tutorialspoint.com/c_standard_library/string_h.htm documents the library

The <stdlib.h> header

- `stdlib.h` defines several functions, including searching, sorting and converting
  - `#include <stdlib.h>` is required to use any of these functions
  - https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm documents the library

- Some of the more commonly used functions:
  - `atoi()`, `atof()`, `atol()`, `atoll()` – parses a string of numeric characters into a number of type **int**, **double**, **long int**, or **long long int**, respectively

Next Lecture

- Pointers