Characters and Strings

Characters

- Characters are the fundamental building blocks of source programs
  - Traditionally, a character ariable is stored in a single byte of memory
  - Each character is encoded as a number using the 7-bit ASCII encoding
  - If you use a different locale, the character may take 2 bytes (think about Chinese characters); this has implications in how you allocate and use memory
- Character constants
  - One character surrounded by *single quotes*
  - `'A'` or `'?'`
  - Since characters are encoded as integers, they allow arithmetic and logical operations; the following are legal:

```
'a' + 2   // Evaluates to 'c'
'a' < 'c' // Evaluates to true
```

- Characters are declared by the keyword `char`
  - Some operations on characters

```
char ch;
for ( ch = 'a'; ch <= 'z'; ch++ )
    printf ( "%c", ch );
ch = getchar(); //read in a character from stdin
```

  - Since a character is an 8-bit byte, it can hold any value from -128 to 127
  - You can use the keyword `unsigned` to keep only positive values
    - `unsigned char ch;` will allow `ch` to hold any value in the range $[0, 255]$
- Special characters and non-graphic characters
  - Denoted by preceding other characters with a backslash `\`

`\n`	newline
`\t`	horizontal tab
`\v`	vertical tab
`\b`	backspace
`\r`	carriage return
`\f`	form feed
`\\`	backslash
`\'`	single quote
`\a`	alert

  - Another form `\ddd` where each `d` is an octal digit
    - `ddd` specifies the desired internal value of a character

  - `NULL` character
    * Indicated by the escape sequence `\0`
    * All bits corresponding to the character are zero
    * Not the same as the ASCII character `0` which is represented by hexadecimal sequence `30`

- String constants
  - Also known as *literals*
  - Sequence of characters surrounded by double quotes
  - Backslash can be used for special characters
  - Double quotes within the string written as `\"`
  - `"ABC"` or `"%d\n\n"`
  - A null character (`\0`) is added immediately after the final character of a string
    * `"ABC"` is stored in four bytes as `"ABC\0"`
    * String constant `"A"` is different from character constant `'A'`
  - String constant `"ABC\nDEF"` is a two-line string
  - ***Important:*** *A character constant is enclosed in single quotes while string constants are enclosed in double quotes*

Fundamentals of strings and characters

- There is no specific string type in C
  - Strings are arrays of `char` supported by library functions
- String is accessed via a pointer to its first character
- String is also viewed as an array of characters, with `'\0'` being used to terminate the string
- A string variable is declared as a pointer to character (or array)

```
char color[] = "blue";
char * color_ptr = color;
```

  - What is the number of bytes reserved for the string in the above cases?
  - You must declare enough space for the string (especially if you intend to increase the size of the string later on)
  - Never forget to account for the `NULL` character when allocating space for strings
- Assigning a string to another variable
  - Since strings are implemented by a pointer to the first element of the character array, they cannot be copied by a simple assignment
  - `color_ptr = color` does not copy the string into `color_ptr` but merely copies the pointer value
  - You may have to use a function such as `strcpy` to actually achieve the copy operation

  - Assuming that `color_ptr` has enough memory allocated, the following function can also achieve the string copy

```
void copy_string ( char * color_ptr, char * const color )
{
    char * in = color;
    char * out = color_ptr;
    while ( *out++ = *in++ );
}
```

- A string can be read by using `scanf` and the `%s` format specifier, but we will resort to using `fgets` to read strings from `stdio` and files, and `sscanf` to read them from within memory
- Be careful about mixing characters and strings, especially when passing them as parameters to functions
- Initializing strings
  - Can be done using either array or pointer notation
  - `char color[] = "blue";`
    - Compiler allocates the space and copies literal into that space
    - This string can be modified
  - `char * color = "blue";`
    - Compiler just creates a pointer to the string literal
    - This string cannot be modified

Character handling library

- Standard library in C to work with characters and strings
- You must include the header file `ctype.h` to use these functions
- In the following table, the type of variable `c` is an `int`, with its value restricted to that in an `unsigned char` (or that of the predefined constant `EOF`)

Character classification macros or Predicates	
`isalpha(c)`	`c` is a letter, [a–z] or [A-Z]
`isupper(c)`	`c` is an uppercase letter, [A-Z]
`islower(c)`	`c` is a lowercase letter, [a-z]
`isdigit(c)`	`c` is a digit [0-9]
`isxdigit(c)`	`c` is a hexadecimal digit, [0-9], [a-f], or [A-F]
`isalnum(c)`	`c` is an alphanumeric character, a letter or a digit
`isspace(c)`	`c` is a whitespace character, `' '`, `'\t'`, `'\r'`, `'\n'`, `'\f'`, or vertical tab
`ispunct(c)`	`c` is a punctuation character (neither control nor alphanumeric)
`isprint(c)`	`c` is a printing character
`iscntrl(c)`	`c` is a control character or `'\b'`
`isascii(c)`	`c` is an ASCII character, code less than `0200`
`isgraph(c)`	`c` is a visible graphic character
Character conversion macros	
`toascii(c)`	Masks `c` with an appropriate value so that `c` is guaranteed to be in the ASCII range `0` through `0x7f`
Character conversion functions	
`toupper(c)`	Convert `c` to its uppercase equivalent
`tolower(c)`	Convert `c` to its lowercase equivalent

- Program to illustrate the character functions – `char_fns.c`

String conversion functions

- Available in general utilities library (`stdlib`)
- Useful to convert a string of digits to integer or floating point values
- In the following table, `str` represents a string (array) and `ptr` represents a pointer to a character

`atof ( str )`	Convert string to double precision number
`strtod ( str, ptr )`	Convert string to double precision number
`atoi ( str )`	Convert string to integer
`atol ( str )`	Convert string to integer
`strtoi ( str, ptr, base )`	Convert string to integer

- A word about the `strxto?` functions
  - If the value of `ptr` is not `(char **)NULL`, a pointer to character terminating the scan is returned to the location pointed to by `ptr`
  - If no number can be formed, `*ptr` is set to `str` and 0.0 is returned
  - `base` is of type `int` and, if its value is between 0 and 36, is used as the base for conversion
    * `0x` or `0X` are ignored if `base` is 16

Standard I/O library functions

- Require the file `<stdio.h>` to be included

- These functions are: `getchar`, `gets`, `putchar`, `puts`, `sprintf`, `sscanf`
- Writing strings using `printf` and `puts`
  - Use `%s` conversion in `printf` to write strings
  - If the `NULL` character is missing, `printf` continues printing until it finds a `NULL` somewhere in memory
  - Use the conversion `%.ps` to print a part of the string

    ```
    char str[] = "Hello world";
    printf ( "First five characters are: %.5s\n", str );
    ```

  - Elimination of `.` will print the string in full, if `p` is less than the string length
  - If string is smaller than `p` characters, it is right justified
  - String can be left justified by using `-`, as in `%-ps`
  - Program `str_print.c`

String manipulation functions

- Require the inclusion of the standard string library `<string.h>`
- These functions operate on null-terminated strings
- These functions do not check for overflow of any receiving strings
- In the following table, `s1` and `s2` represent pointers to character type (or strings)

`strcat ( s1, s2 )`	Appends a copy of strings `s2` to the end of string `s1`
`strncat ( s1, s2, n )`	Appends at most `n` characters from `s2` to `s1`
`strcpy ( s1, s2 )`	Copies `s2` to `s1` until the null character
`strncpy ( s1, s2, n )`	Copies `s2` to `s1` until the null character, or `n` characters
`strdup ( s )`	Duplicates string and returns pointer to the new string
`strlen ( s )`	Number of characters in `s`, not including the `NULL` character

- `strcat`, `strncat`, `strcpy`, and `strncpy` return a pointer to the null-terminated string `s1`
- In `strcpy` and `strncpy`, if the length of target string `s1` is more than the source string `s2`, `s1` is padded with `NULL` characters
- `strdup` automatically allocates space for the new string
- `strdup` returns a `NULL` if it cannot allocate space for the duplicate string

String comparison functions

- These functions return an integer which is

0	if the two strings are equal
> 0	if first string is greater than second string (in alphabetic order)
< 0	if first string is smaller than second string (in alphabetic order)

`strcmp ( s1, s2 )`	Lexicographically compare strings `s1` and `s2`
`strncmp ( s1, s2, n )`	Lexicographically compare first `n` characters of strings `s1` and `s2`
`strcasecmp ( s1, s2 )`	Same as `strcmp` but ignore case differences
`strncasecmp ( s1, s2, n )`	Same as `strncmp` but ignore case differences

- The functions do not compare characters following the `NULL` character in the strings
- Collating sequences are different in ASCII and EBCDIC

String search functions

`strchr ( s, c )`	Returns a pointer to the first occurrence of character `c` in string `s`
`strrchr ( s, c )`	Returns a pointer to the last occurrence of character `c` in string `s`
`strpbrk ( s1, s2 )`	Return a pointer in s1 to the first occurrence of any character from s2
`strspn ( s1, s2 )`	
`strcspn ( s1, s2 )`	
`strstr ( s1, s2 )`	
`strtok ( s1, s2 )`	

- `strchr` and `strrchr` return a pointer to `NULL` if the character `c` does not appear in the string `s`

Memory functions

- Declared in `<memory.h>` file

`memcpy ( s1, s2, n )`
`memccpy ( s1, s2, c, n )`
`memchr ( s, c, n )`
`memcmp ( s1, s2, n )`
`memset ( s, c, n )`