Practice with Characters and Strings

The primary purpose of this lab exercise is to give you practice in basic C by manipulating characters and strings.

1 The Basics

C provides the `char` data type for declaring character and string variables, as in the following:

```
char c, s[100];
```

As with all arrays in C, the first index of array `s` above is 0; the last index of array `s` above is 99.

Character literals are written as `'<character>'`, where <character> is replaced by:

- a single character, as in `'x'`
- an escape sequence of the form `\` followed by a character, as in `'\n'`
- an escape sequence of the form `\x` followed by one or two hexadecimal digits, as in `'\x1f'`

The most common escape sequences that you will encounter are `'\n'` (newline), `'\t'` (tab), `'\\'` (backslash), `'\''` (single quote), `'\"'` (double quote), and `'\0'` (end of string).

String literals are written as `"<0 or more characters>"`; the quote characters `"` are ***not*** part of the string. Escape sequences can be embedded within string literals, and have the same meaning as for character literals.

Unlike strings in some other languages (e.g. Java), the length of a string is not part of its type. A character array (string) must be terminated by the end of string character, `'\0'`. **Therefore, if a string contains `N` characters, the character array that holds the string must be at least of size `N+1`**; the extra location is needed to hold the end of string character (EOS).

2 Standard library functions

Two sets of library functions that are important to the manipulation of characters and strings are defined in <ctype.h> and <string.h>. The most important functions defined in these header files are discussed below.

2.1 <ctype.h>

The header `<ctype.h>` declares functions for testing characters. For each function, the argument is an `int`, whose value must be EOF or representable as an `unsigned char`, and the return value is an `int`. The functions return non-zero (TRUE) if the argument satisfies the condition described, and zero (FALSE) if not.

`isalnum(c)`	`isalpha(c)` is TRUE or `isdigit(c)` is TRUE
`isalpha(c)`	`isupper(c)` is TRUE or `islower(c)` is TRUE
`iscntrl(c)`	control character (for ASCII, c < 0x20)
`isdigit(c)`	decimal digit
`isgraph(c)`	printable character except `' '` (space)
`islower(c)`	lower-case letter
`isprint(c)`	printable character including `' '` (space)
`ispunct(c)`	printable character except space or letter or digit
`isspace(c)`	space, formfeed, newline, carriage return, tab, vertical tab
`isupper(c)`	upper-case letter
`isxdigit(c)`	hexadecimal digit

Additionally, two functions are defined that will convert the case of letters:

`int tolower(int c);`	convert c to lower case if it is an upper case letter
`int toupper(int c);`	convert c to upper case if it is a lower case letter

In each case, the function returns the letter itself if it is not in the scope of the function – i.e. if `tolower(';')` is invoked, it returns `';'`.

2.2 <string.h>

The names of the functions defined for string manipulation all begin with "str", and are enumerated below. Note that if the function copies from one string to another, the behaviour is undefined if the two strings overlap in memory.

We will not actually use any of these functions other than `strlen()` in the remainder of the lab since many of them return pointers to strings. Note that `size_t` is a defined symbolic constant that is usually mapped to `int`; for all intents and purposes, it is an `int`. We will refer back to this list in future labs.

```
char * strcpy(char s[], const char t[]);
```
copy string `t` to string `s`, including EOS; return `s`

```
char * strncpy(char s[], const char t[], size_t n);
```
copy at most `n` characters from `t` to `s`; return `s`; pad with EOS's if `t` has fewer than `n` characters

```
char * strcat(char s[], const char t[]);
```
concatenate string `t` to end of string `s`; return `s`

```
char * strncat(char s[], const char t[], size_t n);
```
concatenate at most `n` characters of string `t` to string `s`; terminate `s` with EOS; return `s`

```
int strcmp(const char s[], const char t[]);
```
compare `s` to `t`; return <0 if `s` < `t`, 0 if `s` == `t`, >0 if `s` > `t`

```
int strncmp(const char s[], const char t[], size_t n);
```

compare at most `n` characters of `s` to `t`; return <0 if `s` < `t`, 0 if `s == t`, >0 if `s` > `t`

```
char * strchr(const char s[], int c);
```

return pointer to first occurrence of `c` in `s` or `NULL` if not present

```
char * strrchr(const char s[], int c);
```

return pointer to last occurrence of `c` in `s` or `NULL` if not present

```
size_t strspn(const char s[], const char t[]);
```

return length of prefix of `s` consisting of characters in `t`

```
size_t strcspn(const char s[], const char t[]);
```

return length of prefix of `s` consisting of characters ***not*** in `t`

```
char * strpbrk(const char s[], const char t[]);
```

return pointer to first occurrence in string `s` of any character of string `t`, or `NULL` if none are present

```
char * strstr(const char s[], const char t[]);
```

return pointer to first occurrence of `t` in `s`, or `NULL` if not present

```
size_t strlen(const char s[]);
```

return length of `s`

```
char * strerror(int n);
```

return pointer to implementation-defined string corresponding to error `n`

```
char * strtok(char s[], const char t[]);
```

searches `s` for tokens delimited by characters from `t`

3 Exercises

Several programs that manipulate strings are specified below. You should design, create, compile, and test each program. Each builds upon the other, so it is not as imposing a task as you might think.

Recall that if your source code is in `file.c`, then the appropriate command to compile and link your program is as follows (the '%' character simply indicates the prompt from the shell.

```
% clang –Wall -Werror –o file file.c
```

If the compile and link was successful, then you execute your program with the following command:

```
% ./file
```

Remember that if you wish to redirect standard input or standard output, you need to use the io redirection syntax for the shell, such as

```
% ./file < file.c
```

This will cause your program to operate on the contents of `file.c`, as opposed to waiting for you to type in the data to be processed.

3.1 File copy character at a time

We saw today in the labs the use of `scanf` and `printf` to read and write characters one at a time. Create a file named `repeat.c` that prints out ever character that is read from standard input, compile it, and test it.

3.2 File copy line at a time

Since we have not yet completed our discussion of pointers, I will provide you with the following two functions to use in the remainder of the lab to read and write a line at a time. The function prototypes are:

```
int readline(char line[], int max);
int writeline(const char line[]);
```

`readline` reads the next input line (including the newline) from standard input into the character array `line`; at most `max-1` characters will be read. The resulting line is terminated with an `EOS`. The function return is the number of characters copied into line. If `readline` returns 0, this indicates the end of file.

`writeline` writes `line` to the standard output, returning the number of characters written.

The source code for these two functions is as follows:

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

/* readline: read a line from standard input, return its length or 0
*/
int readline(char line[], int max)
```

```
{
   if (fgets(line, max, stdin) == NULL)
      return 0;
   else
      return strlen(line);
}

/* writeline: write line to standard output, return number of chars
written */
int writeline(const char line[])
{
   fputs(line, stdout);
   return strlen(line);
}
```

Create a file named `lfcopy.c` that copies standard input to standard output a line at a time using `readline` and `writeline`, compile it, and test it.

3.3 Line counting

Write a program for counting lines on standard input reading a character at a time. Wrtie a second program that counts lines using the `readline` introduced in section 3.2 above and prints the result on standard output. Compile and test both programs[1].

3.4 Real word counting

Write a program that counts words, where a word is defined as a sequence of characters that are all letters – i.e. no digits, punctuation marks, white space, control characters, etc. Create a file named `wcount.c` that counts the number of words using this definition, and prints the results on standard output. Compile and test the program. (*Hint: you may want to use one or more of the functions defined in ctype.h to help you with this exercise.*)

3.5 Collapsing White Space

Write a program that copies standard input to standard output, converting all sequences of white space (as defined by isspace()) to a single space. Since `'\n'` is considered white space by isspace(), you will have to be sure to write a newline character to standard output before you exit your program. What happens when the input contains quoted strings containing sequences of two or more white space characters? Is this correct behavior? How would you go about fixing this in your program?

3.6 Questions taken from Google Interviews

- Write a function `f(a, b)` which takes two character string arguments and returns a string containing only the characters found in both strings in the

[1] Linux provides the `wc` command to count characters, lines, words, … in files. If invoked using the following command syntax:

```
% wc -l <file
```

it will perform the identical function to your program. Therefore, you can test `lcount` by comparing its output to that from `wc`.

order of `a`.
Write a version which is order N-squared and one which is order N.

- There is an array `A[N]` of `N` numbers.
  You have to compose an array `Output[N]` such that `Output[i]` will be equal to multiplication of all the elements of `A[N]` except `A[i]`.
  For example `Output[0]` will be multiplication of `A[1]` to `A[N-1]` and `Output[1]` will be multiplication of `A[0]` and from `A[2]` to `A[N-1]`.
  Solve it without division operator and in O(N).