Processing Text and Files
(using Standard C Library functions)

---

1. Consider the following program the exemplifies the use of the `main` function in its most general form in C.

```
#include <stdio.h>

int main (int argc, char* argv[]) {
  printf("# arguments = %d\n", argc - 1);
  printf("the command arguments are: %s\n", argv[0]);
  for (int i = 1; i < argc ; i++)
    printf("argv[%d]=%s\n", i, argv[i]);
  return 0;
}
```

Compile the program and try it with the following commands:

```
$ gcc -Wall maintest.c -o maintest
$ ./maintest
$ ./maintest  mercury
$ ./maintest  mercury venus
$ ./maintest  mercury venus earth
$ ./maintest  mercury venus earth mars
```

The type of the `main` function is:

```
int main (int argc, char* argv[])
```

Here `argc` is the number of strings in the command line and `argv` is a vector that stores all those strings (e.g., in the second example, `argv[0] = "./maintest"` and `argv[1] = "mercury"`). This form of the `main` function is very useful as it allows the programmer to pass values to the application without using I/O functions such as `scanf`.

2. Consider the following program that takes two strings from the command line (`argv[1]` e `argv[2]`) and manipulates them with the *string* subset of the Standard C Library (`clib`) API. Compile it and try it.

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

int main(int argc, char* argv[]) {
  /* check if you have the right number of arguments */
  if ( argc != 3 ) {
    printf("usage: %s string1 string2\n, argv[0]);
    exit(EXIT_FAILURE);
  }
  /* compare argv[1] and argv[2] using lexicographic order */
  int result = strcmp(argv[1], argv[2]);
  if (result == 0)
    printf("the strings are the same\n");
  else if (result < 0)
    printf("%s < %s\n", argv[1], argv[2]);
  else
    printf("%s > %s\n", argv[1], argv[2]);

  /* create a copy of argv[1] and another of argv[2] */
  char *p1 = strdup(argv[1]);
  char *p2 = strdup(argv[2]);
  printf("p1 holds:%s\n", p1);
  printf("p2 holds:%s\n", p2);

  /* * this is another way of doing it */
  char* p3 = (char*)malloc((strlen(argv[1]) + 1) * sizeof(char));
  char* p4 = (char*)malloc((strlen(argv[2]) + 1) * sizeof(char));
  strcpy(p3, argv[1]);
  strcpy(p4, argv[2]);
  printf("p3 holds:%s\n", p3);
  printf("p4 holds:%s\n", p4);

  /* concatenate both strings, allocating space for:
        all chars of argv[1],
        all chars of argv[2],
        the final '\0'  */
  char* p5 = (char*)malloc((strlen(argv[1]) + strlen(argv[2]) + 1) * sizeof(char));
  strcpy(p5, p1);
```

```
  strcat(p5, p2);
  printf("p5 holds:%s\n", p5);

  /* free allocated memory from malloc and strdup */
  free(p1);
  free(p2);
  free(p3);
  free(p4);
  free(p5);
  exit(EXIT_SUCCESS);
}
```

Run the command `man 3 string` to see the full set of functions in this API. Based on this example, write a program that:

- gets a string from the command line and transforms it into an equivalent string but in lowercase;
- gets two strings from the command line and indicates whether the first occurs within the second;
- gets two strings from the command line and indicates how many times the first occurs in the second.

Suggestion: do `man tolower` and `man toupper` to see `clib` functions that may be relevant.

3. Consider the following program that opens a file whose name is given as an argument (`argv[1]` in the code), reads its content in blocks of `BUFFER_SIZE` bytes and that writes those bytes to the terminal (`stdout`).

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

#define BUFFER_SIZE 1024

int main(int argc, char* argv[]) {
  FILE*   file = fopen(argv[1], "r");
  if ( file == NULL ) {
     printf("error: could not open %s\n", argv[1]);
     exit(EXIT_FAILURE);
  }
  char buffer[BUFFER_SIZE];
  int  nchars = fread(buffer, sizeof(char), BUFFER_SIZE, file);
```

```
  while (nchars > 0) {
     fwrite(buffer, sizeof(char), nchars, stdout);
     nchars = fread(buffer, sizeof(char), BUFFER_SIZE, file);
  }
  fclose(file);
  exit(EXIT_SUCCESS);
}
```

Compile the program and try it with the following commands:

```
$ gcc -Wall filetest.c -o filetest
$ cat > quote.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras gravida nisl tortor, eget vulputate lacus viverra non.
Proin pharetra gravida condimentum.
Nam imperdiet dictum placerat.
^D
$ ./filetest quote.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras gravida nisl tortor, eget vulputate lacus viverra non.
Proin pharetra gravida condimentum.
Nam imperdiet dictum placerat.
$
```

Check the system manual pages for the functions `fopen`, `fread`, `fwrite` and `fclose` and try to understand how the program works. What happens if you define `BUFFER_SIZE` with the value 1?

4. Based on the previous exercise, write a command called `mycat` that:

- receives the name of a file as an argument and prints its content (this is exactly what the shell command `cat` when given a file name);
- receives a sequence of file names as arguments and prints their contents in the terminal sequentially (again, this is similar to the way `cat` works).

```
$ gcc -Wall mycat.c -o mycat
$ cat > file1
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras gravida nisl tortor, eget vulputate lacus viverra non.
Proin pharetra gravida condimentum.
Nam imperdiet dictum placerat.
^D
$ cat > file2
```

```
Sed convallis hendrerit scelerisque.
Sed sodales sagittis nulla vitae auctor.
Quisque lobortis tortor vitae ligula ullamcorper fermentum.
Aliquam interdum, metus sed rhoncus gravida,
nibh nisl porttitor tortor, in finibus mauris erat et lacus.
^D
$ cat > file3
Aliquam sit amet arcu molestie, sodales sem vitae, semper nisi.
Curabitur lacinia vel metus in aliquam.
Fusce non tellus pulvinar, tincidunt quam ac, rhoncus turpis.
^D
$ ./mycat file1
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras gravida nisl tortor, eget vulputate lacus viverra non.
Proin pharetra gravida condimentum.
Nam imperdiet dictum placerat.
$ ./mycat file1 file2 file3
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras gravida nisl tortor, eget vulputate lacus viverra non.
Proin pharetra gravida condimentum.
Nam imperdiet dictum placerat.
Sed convallis hendrerit scelerisque.
Sed sodales sagittis nulla vitae auctor.
Quisque lobortis tortor vitae ligula ullamcorper fermentum.
Aliquam interdum, metus sed rhoncus gravida,
nibh nisl porttitor tortor, in finibus mauris erat et lacus.
Aliquam sit amet arcu molestie, sodales sem vitae, semper nisi.
Curabitur lacinia vel metus in aliquam.
Fusce non tellus pulvinar, tincidunt quam ac, rhoncus turpis.
```

5. Write a program that receives two file names as arguments (`argv[1]` and `argv[2]` in the code) and that copies the content of the first file to the second file. If the second file does not exist, then it must be created. If it already exists, its contents will be overwritten. This emulates the behavior of the Bash shell command `cp`.

```
$ gcc -Wall mycp.c -o mycp
$ cat > file1
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
^D
$ ./mycp file1 file2
$ cat file2
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
$ cat > file3
```

```
Cras gravida nisl tortor, eget vulputate lacus viverra non.
^D
$ ./mycp file3 file2
$ cat file2
Cras gravida nisl tortor, eget vulputate lacus viverra non.
```

6. Sometimes you want to process a text file and read it line-by-line. For this purpose, the function `getline` is more robust and safe than its siblings in the `libc`. When it receives a `NULL` buffer pointer, it allocates the buffer internally and returns with the line just read. The buffer pointer will also be initialized. Here is an example of its use. Check the manual page to learn more.

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

int main(int argc, char *argv[]) {
  /* open file, exit on error */
  FILE*   file = fopen(argv[1], "r");
  if ( file == NULL ) {
    printf("error: could not open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }
  /* read file, line by line */
  ssize_t read;
  size_t  size = 0;
  char*   line = NULL;
  int lineno = 1;
  while ((read = getline(&line, &size, file)) != -1) {
    printf("[%5d]: %s", lineno, line);
    lineno++;
  }
  /* close file */
  fclose(file);
  /* return gracefully */
  exit(EXIT_SUCCESS);
}
```

7. Write a program `mywc` that, given a text file as a command line argument, prints:

- the number of characters in the file, if an option `-c` is used;
- the number of words in the file, if an option `-w` is used;

- the number of lines in the file, if an option `-l` is used.

```
$ gcc -Wall mywc.c -o mywc
$ cat > file.txt
This is a test
^D
$ ./mywc -c file.txt
15
$ ./mywc -w file.txt
4
$ ./mywc -l file.txt
1
```

Suggestion: the function `getopt` from the `libc` will make it easier to manage the command line options. Compare your program to the shell command `wc`.

8. Sometimes it is useful to break a text line in its constituent words. The `libc` has two functions that can perform this operation: `strtok` e `strsep`. Of these two, `strsep` is more robust. You can see a simple example of its use in the following code:

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

int main(int argc, char* argv[]) {
  int   count = 0;
  char* text  = strdup(argv[1]);
  char* word;
  while ((word = strsep(&text, " \t")) != NULL) {
    if ( *word == '\0' ) /* skip multiple occurrences of delimiter */
       continue;
    printf("%s\n", word);
    count++;
  }
  printf("%d words found\n", count);
  exit(EXIT_SUCCESS);
}
```

9. Write a program `mygrep` that, given a string and a file from the command line, prints all occurrences of the string in the file, indicating the line and column where these begin. The output would look something like:

```
$ gcc -Wall mygrep.c -o mygrep
```

```
$ ./mygrep needle haystack.txt
[2:17]
[5:2]
[23:7]
```

10. Write a program `findrepl.c` that takes a list of pairs of words from the command line (in the form of `findword-replword`) and text input from the `stdin` and replaces the `findword` with `replword`, writing the output to `stdout`. Note that, for a pair `sword-FlOwer` the words to be replaced must exactly match `sword` and that it must be replaced with `FlOwer` exactly. To compile and test the program you would do something like:

```
$ gcc -Wall findrepl.c -o findrepl
$ cat > sometext.txt
Flower, stone, Grass.
Riffle, Sword, spear.
Water, Stone, fire, Air.
No sword, no riffle, no spear.
^D
$ ./findrepl sword-FlOwer stone-Earth < sometext.txt
Flower, Earth, Grass.
Riffle, Sword, spear.
Water, Stone, fire, Air.
No FlOwer, no riffle, no spear.
```