Processamento de Texto e Ficheiros (usando funções da Standard C Library)

---

1. Considere o seguinte prograna que exemplifica o uso da função `main` na sua forma mais geral em 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 o programa com o comando:

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

e experimente-o com os comandos:

```
$ ./maintest
$ ./maintest  mercury
$ ./maintest  mercury venus
$ ./maintest  mercury venus earth
$ ./maintest  mercury venus earth mars
```

O tipo da função `main` é:

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

Em que `argc` é o número de "strings"na linha de comando e `argv` é um vector com todas essas "strings"(e.g., no segundo exemplo, `argv[0] = "./maintest"` e `argv[1] = "mercury"`. Esta forma da função `main` é muito útil pois permite passar valores inicialmente para a aplicação sem precisarmos de usar funções de I/O como o `scanf`.

2. Considere o seguinte programa que recebe duas strings na linha de comando (`argv[1]` e `argv[2]`) e realiza operações com elas com a API de *strings* da Biblioteca Standard do C (`clib`). Compile-o e experimente-o.

```
#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);
}
```

Faça `man 3 string` para ver a API completa. Com base neste exemplo, escreva agora um programa que:

- recebe uma string na linha de comando e a transforma numa string equivalente mas com todos os caracteres em minúsculas;
- recebe duas strings na linha de comando e indica se a primeira ocorre na segunda;
- recebe duas strings na linha de comando e indica quantas vezes a primeira ocorre na segunda.

Sugestão para a primeira alínea: faça `man tolower` e `man toupper` para ver funções da `clib` que podem ser relevantes.

3. Considere o seguinte programa que abre um ficheiro (cujo nome é passado como um argumento do programa, `argv[1]` no código), lê o seu conteúdo em blocos de `BUFFER_SIZE` bytes de cada vez e escreve esses bytes para o 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 o programa e experimente-o com os seguintes comandos:

```
$ 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.
$
```

Consulte a página de manual das funções `fopen`, `fread`, `fwrite` e `fclose` e tente compreender como o programa funciona. Que acontece se definir `BUFFER_SIZE` com 1?

4. Com base no programa anterior, escreva um comando `mycat` que:

- recebe como argumento o nome de um ficheiro e imprime o seu conteúdo (semelhante ao comando `cat` com 1 argumento);
- recebe como argumento os nomes de vários ficheiros e imprime o conteúdo de todos eles sequencialmente (semelhante ao comando `cat` com vários argumentos).

```
$ 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. Escreva um programa que receba o nome de dois ficheiros como argumentos, em `argv[1]` e `argv[2]`, e copie o conteúdo do primeiro ficheiro para o segundo. Se o segundo ficheiro não existir deverá ser criado. Se o ficheiro existir o seu conteúdo será reescrito. Esta é a forma como funciona o comando `cp` da Bash shell.

```
$ 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. Por vezes, queremos processar um ficheiro de texto lendo-o linha por linha. Para este propósito, a função `getline` é mais robusta e segura do que as suas congéneres na `libc`. Quando invocada com um apontador `NULL` no local onde deveria estar o endereço do buffer (`line`) onde a linha do ficheiro deveria ser escrita, a função faz ela própria a reserva desse espaço no heap e retorna com a linha copiada para essa localização e o apontador inicializado. Aqui está um exemplo simples da sua utilização. Leia o manual para sabere mais detalhes.

```
#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. Escreva um programa `mywc` que dado um ficheiro de texto como argumento escreva:

- o número de caracteres, se a opção fôr `-c`

- o número de palavras, se a opção fôr `-w`
- o número de linhas, se a opção fôr `-l`

```
$ gcc -Wall mywc.c -o mywc
$ cat > file.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
^D
$ ./mywc -c file.txt
57
$ ./mywc -w file.txt
8
$ ./mywc -l file.txt
1
```

Sugestão: a função `getopt` da `libc` facilitará a gestão das opções da linha de comandos. Compare o seu programa com o programa homónimo da Bash shell.

8. Por vezes é útil podermos dividir uma linha de texto nas palavras que a constituem. A `libc` tem duas funções que permitem fazer este tipo de processamento: `strtok` e `strsep`. A função `strsep` é a mais robusta das duas. Pode ver um exemplo simples da sua utilização no seguinte código:

```
#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. Escreva um programa `mygrep` que dada uma string e o nome de um ficheiro na linha de comando imprima todas as ocorrências da string no ficheiro, indicando a linha e a coluna

do texto onde começam. O modo de execução do comando e o respectivo "output" devem seguir o seguinte padrão:

```
$ gcc -Wall mygrep.c -o mygrep
$ ./mygrep string file
[2:17, 5:2, 23:7]
```

10. Escreva um programa `findrepl.c` que dada uma lista de pares de palavras na linha de comando (na forma `findword-replword`) e um texto proveniente do `stdin` encontre e substitua as ocorrências de cada `findword` por `replword`, escrevendo o texto resultante para o `stdout`. Note que para um par `sword-FlOwer` as palavras a serem substituídas devem ser exactamente iguais a `sword` e devem ser substituídas exactamente por `FlOwer`. Para compilar e testar o programa pode fazer algo como:

```
$ 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.
```