

CSE1301 Exercise Sheet 7 Strings and File I/O

Exercise 1

Write a C expression which is true only when the character string stored in the variable `surname` begins with a letter between 'A' and 'M' i.e. excludes 'A' and 'M'.

Exercise 2

Write the C code to

- (a) open a file for an input stream;
- (b) open a file for an output stream.

Exercise 3

Suppose that your name (first name followed by family name) is in a file called `name.txt`. Write a C program to read in your name, and then print it to a new file with your family name followed by your first name.

Example input file `name.txt`: John Smith

Example output file `newname.txt`: Smith, John

Exercise 4

Write fragments of C code showing how you might use the string library functions `strcmp()`, `strcpy()`, and `strlen()`. Then write an algorithm for a `strcpy()` function, be clear on the inputs and what it returns.

Exercise 5

Assume a file contains a coded message in the form of a list of letters and spaces terminated by the end marker 9 or 8. Write an algorithm to decode the message by writing every fourth character to an output file, `message.txt`, and also display the original coded message on the screen. Code your algorithm in C.

Exercise 6

Compression programs take a file of text and produce another file which encodes the same text but is shorter. A simple form of compression involves assigning short tokens to character sequences (i.e., words) which occur more than once in the text and replacing those words by their token wherever they appear in the text. For example:

"the cat was on the mat, so the mat was not on the cat - as the cat was on it."
could be encoded as:

"1the2cat3was4on5mat 1 2 3 4 1 5, so 1 5 3 not 4 1 2 - as 1 2 3 4 it."

thereby saving 10 characters. Note that punctuation, spacing, and words occurring only once are preserved. Extended spacing could also be encoded by treating multiple blanks as special words.

The *compression factor* for a particular compression is the number of characters saved by encoding, divided by the number of characters in the original. So for the above example the compression factor is $10/77$ or 13%

Write a program to encode text files in this fashion. Under what conditions would this compression technique be likely to produce an encoded text shorter than the original? Could it produce a longer encoded text?

Add some code which determines *before the encoding phase* whether the encoded version will be shorter or longer than the original and hence predicts the compression factor. Use the predicted compression factor as the basis for deciding whether or not to encode the text.

The length of the tokens used to encode the words is critical to the success of compression. In particular, the most frequently used words should have the shortest tokens. Why?

Modify your program to take this into account. Do your compression factors improve?