University of Exeter

Faculty of Environment, Science

and Economy

ECM2433

The C Family

Continuous Assessment

Date Set:	1st February 2024
Date Due:	22nd February 2024
Return Date:	14th March 2024

This CA comprises 40% of the overall module assessment.

This is an **individual** exercise and your attention is drawn to the Faculty and University guidelines on collaboration and plagiarism, which are available from the Faculty website.

This is the only coursework for this module and tests your skills and understanding of programming in C.

Please Turn Over

Programming excercise

For this coursework you must write an ANSI standard C program, whose executable is named `coursework`, that performs the following three tasks:

Task 1: Read memory in bytes

Write a function called `printBytes` that accepts a void pointer called `ptr` and an integer called `numBytes` as inputs. This function will then print out the decimal values stored in the `numBytes` contiguous memory locations starting at the memory address pointed to by `ptr`. If, for example, the function is called like this:

```
printBytes(somePointer,4)
```

where `somePointer` is a pointer value, and the four bytes stored at that address and the three following bytes are 10, 20, 30 and 145, then the output will look like this:

```
Starting at memory address XXXXXX:
001:  10
002:  20
003:  30
004: 145
```

where the "`XXXXXX`" is replaced with the memory address stored in `somePointer`.

Task 2: Read data from one file and output it to another

The program must be run from the Linux command line like this:

```
coursework fileIn fileOut
```

where `fileIn` is the name of an existing text file to be read in, and `fileOut` is the name of the file to be output to.

The input file will contain one line of text of unknown length. Your program must read this in from the `fileIn` file and then write it out to the `fileOut` file in reverse order. So, if, for example, `fileIn` contains the text "I love C!", then when the program has completed, `fileOut` must contain "!C evol I" (all on one line).

Task 3: Create a new string datatype

On the ELE page you will find a C header file called `msString.h` which contains the definition of a datatype called `msString` and function prototypes for a number of functions.

As you know, C stores a string as an array of characters, terminated by an extra, null character. You are going to define a new way of storing a string. This new `msString` data type is a pointer, which points to a memory address that stores a `long int` value that is the length of the string (the number of characters), followed by the characters that make up the string (with no terminating null character).

The functions that support this new datatype are as follows:

msSetString: accepts a standard C string as an input and returns a new `msString`.

msGetString: accepts an `msString` as an input and returns a standard C string.

msCopy: accepts two inputs, a pointer to an `msString` (the destination) and an `msString` (the source), and copies the source value to the destination.

msConcatenate: accepts two inputs, a pointer to an `msString` (the destination) and an `msString` (the source), and concatenates the source value onto the end of the destination.

msLength: accepts an `msString` as an input and returns an integer value that is the number of characters in the string.

msCompare: accepts two `msString` values as inputs and returns 0 (zero) if they are identical, or 1 if they are not.

msCompareString: accepts an `msString` value and a standard C string value as inputs and returns 0 (zero) if the characters making up the strings are the same, or 1 if they are not.

msError: accepts a standard C string as input, outputs it as an error message to the user and then exits the program. This function can be called by any of the other `msString` functions when an error occurs.

You must write a C module that implements these functions, without changing the prototypes or definitions. You can then test your coding by using the following C code in your `coursework.c` program:

```
1  msString   ms     = msSetString("Hello");
2  msString   ms2    = msSetString(" World!");
3  msString   mscopy = NULL;
4
5  printf("String |%s| is %d characters long (%p).\n",
6         msGetString(ms),msLength(ms),ms);
7  msCopy(&mscopy,ms);
8  printf("Copied string |%s| is %d characters long (%p).\n",
9         msGetString(mscopy),msLength(mscopy),mscopy);
10
11 printf("Compare ms with mscopy: %d\n",msCompare(ms,mscopy));
12 printf("Compare ms with ms2   : %d\n",msCompare(ms,ms2));
13 printf("Compare ms with Hello : %d\n",msCompareString(ms,"Hello"));
14 printf("Compare ms with HelloX: %d\n",msCompareString(ms,"HelloX"));
15 printf("Compare ms with Hella : %d\n",msCompareString(ms,"Hella"));
16
17 msConcatenate(&mscopy,ms2);
18 printf("Concatenated string |%s| is %d characters long (%p).\n",
19        msGetString(mscopy),msLength(mscopy),mscopy);
```

You will need to add further code around this to make it a well-structured C program, and you will need to change it to remove the memory leak (mark this with an explicit comment in the code). You may also choose to add further tests. The output from this code should be like this:

```
1  String |Hello| is 5 characters long (0x2304010).
2  Copied string |Hello| is 5 characters long (0x2304070).
3  Compare ms with mscopy: 0
4  Compare ms with ms2   : 1
5  Compare ms with Hello : 0
6  Compare ms with HelloX: 1
7  Compare ms with Hella : 1
8  Concatenated string |Hello World!| is 12 characters long (0x2304070).
```

where the values within brackets are memory addresses that may change each time you run it. Note however that the address printed in line 1 is different from that in line 2, but that lines 2 and 8 show the same memory address.

Deliverables

The deliverables for this coursework comprise all the source code for your program, and a Linux shell script (called `compile`) for compiling and linking it into a single executable, which **must** be called `coursework`. These should all be zipped together and submitted via ELE.

Your program must compile, link and execute on one of the two Linux servers we are using on this module (`emps-ugcs1` and `emps-ugcs2`).

Submission must be made by 12pm (noon) on the date indicated on the front page of this document.

Marking Scheme

Criteria	**Marks**
Program basics *To what extent is your code well-constructed, well-formatted, and to the ANSI standard? Does your program compile, link and run without errors and warnings? To what extent does it trap and report run-time errors?*	30
Task 1: Read and print bytes of memory (`printBytes` function) *To what extent does the program correctly and efficiently print out the values stored at the memory locations?*	10
Task 2: File read/write *To what extent does the program correctly and efficiently read in the command line parameters, read the contents of the input file and write the new content to the output file?*	10
Task 3: Implementation of the new `msString` functions *To what extent does your program correctly and efficiently implement the functions prototyped in the supplied `msString.h` C header file? Have you correctly identified the memory leak in the provided test code?*	50
Total	100