The University of Western Australia

2nd SEMESTER EXAMINATIONS
NOVEMBER 2020

**DEPARTMENT OF COMPUTER SCIENCE &
SOFTWARE ENGINEERING**

CITS2002 SYSTEMS PROGRAMMING

This Paper Contains 6 Pages and 4 Questions
You are required to attempt ALL FOUR (4) questions.

Time Allowed: 2 hours

PLEASE NOTE

You may use any number of blank pages for rough working. You do not need to submit these pages at the end of the exam.

No other materials, such as textbooks, notes, or calculators, are permitted during the exam.

Students using ExamSoft should answer all questions using "Courier New" (fixed-width) font. This can be selected at the top left corner of the answer field."

THIS PAGE INTENTIONALLY LEFT BLANK

1) In the C99 programming language, a colour can be represented by 3 integer values, each between 0 and 255, storing the magnitude of the colour's red, green, and blue components.

One measure of the *distance* between any two colours, is the *sum of the squares of the differences* between their red, green, and blue components. For example, if the Colour1 is represented by the integers R1, G1, and B1, and Colour2 is represented by the integers R2, G2, and B2, then the *distance* between the two colours may be expressed as:

```
distance = (R2-R1)*(R2-R1) + (G2-G1)*(G2-G1) + (B2-B1)*(B2-B1);
```

The smaller the distance, the closer are the two colours.

Consider the C99 function whose prototype is:

```
char *closestRGB(char *filename, int R1, int G1, int B1);
```

The function receives 4 parameters – the name of a text file containing many colours (one per line), and three integers. The text file might typically contain:

```
190 190 190   grey
 65 105 225   RoyalBlue
  0 255 255   cyan
  0 100   0   DarkGreen
```

where each line provides the red, green, and blue components of the named colour. Each line's fields are separated by whitespace characters.

The purpose of the function is to return a pointer to dynamically allocated memory storing the *name* of the colour *closest* to the colour represented by the function's integer parameters. If the function detects any problems, the function should simply return NULL.

Write the `closestRGB()` function in C99.

(10)

---

2) One data-structure that has grown in prominence in newer programming languages is the *hashtable* (or hashmap, or dictionary). They are similar to arrays, in that we use an index to add or find a required item. However, unlike the standard arrays in C99 where the index is an integer, a hashtable's index may be of another datatype, such as a string.

Unlike some other programming languages, C99 does not provide a hashtable data-structure, but one can be implemented using a few simple functions that manage dynamically allocated memory.

Assuming that you have a function that hashes a string to an integer, with the prototype:

```
unsigned int string_hash(char *string);
```

develop a simple hashtable data-structure in C99.

Firstly, using C99's **`typedef`** feature, define a user-defined datatype named `HASHTABLE` to represent your data-structure. Your data datatype should employ a fixed sized array of pointers, each of which points to a list.

Each list consists of linked structures, with each structure storing a value (here, a string), the number of times that value has been added to the hashtable, and a pointer to a structure of the same datatype. Each linked list is terminated by a NULL-pointer.

Your data-structure should be managed and accessed by two functions, with the following C99 prototypes:

```
int add_to_hashtable(HASHTABLE *hashtable, char *string);
```

and

```
int find_in_hashtable(HASHTABLE *hashtable, char *string);
```

On success each function returns the number of times that the indicated string is (now) stored in the hashtable, or -1 if the function detects any problem with its parameters or during its execution.

Write `add_to_hashtable()` and `find_in_hashtable()` in C99.

(10)

---

3) *make* is a system utility commonly used to build programming projects. *make* first determines the sequence of commands to be executed to build a project. The commands are then executed, in turn, until all commands have executed successfully, or until one of them is unsuccessful.

make has no knowledge of the commands (and their arguments) that require executing. Instead, *make* invokes the standard shell to execute each command. For example, if *make* determines that an action involving the compilation of a C99 program must be performed, then *make* may execute a command such as:

```
/bin/sh  −c  "cc −o testing.o testing.c"
```

by passing the required command (action) as a single string to the shell, and then capturing the exit status of the shell to determine if the command was successful, or not.

Consider the C99 function whose prototype is:

```
int executeCommandSequence( char *commands[] );
```

The function receives a vector of character strings, each of which is passed to a new shell process to be executed. The vector of strings is terminated by a NULL pointer.

The commands are executed, in turn, until all commands have executed successfully (in which case the function returns 0), or until one of them is unsuccessful (in which case the function returns the exit status of the unsuccessful command).

Write the `executeCommandSequence()` function in C99.

(10)

4a) Explain what information must be managed *within* a Unix-based operating system kernel when a process invokes a `fork()` system-call followed shortly thereafter by a `wait()` system-call, and the new child process invokes an `execve()` system-call and *eventually* invokes an `exit()` system-call.

(5)

4b) When computer systems first supported multi-programming, the memory required for each process was initially allocated using *equal-sized memory partitioning*. While successful, this approach exhibited some problems that were later overcome with the introduction of *unequal-sized memory partitioning.*

i) Briefly describe two problems resulting from the use of fixed-sized partitioning, and describe how these were addressed by using unequal-sized partitions.

After several years of experimentation with different memory partitioning approaches, contemporary computer systems now almost exclusively employ a fixed-sized memory partitioning scheme termed *paging*.

ii) Briefly explain how *paging* is able to overcome the disadvantages of both traditional fixed-sized partitioning and unequal-sized partitioning.

(5)

END OF PAPER