# NWEN 241 Assignment 2

(Weeks 3–5 Topics)

Release Date: **1 May 2020**

Submission Deadline: **18 May 2020, 23:59**

This assignment is divided into 2 parts.

- In Part I (Tasks 1–3), you will be asked to answer questions about Weeks 3–5 topics, submitted in a plain text file named `part1.txt`
- In Part II (Tasks 4–8), you will be asked to implement C functions, the specifications of which are presented in each of the tasks. To be submitted in file named `dbms.c`

You must submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Assignment_2`) on or before the submission deadline. Late submissions (up to 48 hours from the submission deadline) will be accepted but will be penalized. No submissions will be accepted 48 hours after the submission deadline.

Full marks is 100. The following table shows the marks distribution:

| Task Type | Part I | Part II | **Total** |
|---|---|---|---|
| Core | 20 | 45 | **65** |
| Completion | 8 | 12 | **20** |
| Challenge | 5 | 10 | **15** |
| Total | **33** | **67** | **100** |

## Part I: Concepts

This part will test your conceptual knowledge of strings, arrays, pointers, and memory in C. Your answers should be submitted in a plain text file named `part1.txt`.

### Task 1.

**Core [20 Marks]**

1) **[2 Marks]** Declare a pointer to a floating-point value with the identifier `fp`.

2) **[2 Marks]** Declare a prototype for a function `func1` that accepts two integer arguments and returns a pointer to a long integer.

3) **[2 Marks]** Declare a one-dimensional array of integer pointers with 20 elements.

4) **[2 Marks]** Declare an array of strings whose initial values are `"cyan"`, `"magenta"`, and `"yellow"`.

5) **[2 Marks]** Define a structure that can represent the dimensions of a rectangle, with tag `rect`, and consisting of 2 float members `width` and `length`.

6) **[2 Marks]** Use typedef to define a new type `rect_t` from the structure defined in (6).

7) **[4 Marks]** Given the following declaration where `rect_t` is the type defined in (7), write a C statement that will allocate an array of 10 `rect_t` elements, and let `p` point to that memory.

```
    rect_t *p;
```

8) **[4 Marks]** Given the following function prototype, write the function definition that will allocate memory for a `rect_t`, set the width and length to `w` and `l`, respectively, and return a pointer to the allocated memory.

```
    rect_t *create_rect(float w, float l);
```

**Task 2.**

**Completion [8 Marks]**

A C program contains the following statements:

```
1   char u, v = 'A';
2   char *pu, *pv = &v;
3
4   *pv = v + 1;
5   u = *pv + 1;
6   pu = &u;
```

Each character occupies 1 byte of memory. Suppose the value assigned to `u` is stored in (decimal) address `1100` and the value assigned to `v` is stored in (decimal) address `1101`.

1) **[2 Marks]** What is the numeric value of the expression `&u`?

2) **[2 Marks]** What is the numeric value of the expression `&v`?

3) **[2 Marks]** What is the numeric value of the expression `*pv` after the completion of line 4?

4) **[2 Marks]** What is the numeric value of the expression `*pu` after the completion of line 6?

**Task 3.**

**Challenge [5 Marks]**

A C program contains the following statements:

```
    short a[] = {1, 2, 4, 8, 16, 32};
    short *pa = a;
    short **ppa = &pa;
```

Suppose each short integer quantity occupies 2 bytes of memory, the array `a` is at (decimal) address 1102, `pa` is at (decimal) address 1114, and `ppa` is at (decimal) address 1118.

1) **[1 Mark]** What is the numeric value of the expression `a[0]`?

2) **[1 Mark]** What is the numeric value of the expression `&a[0]`?

3) **[1 Mark]** What is the numeric value of the expression `a`?

4) **[1 Mark]** What is the numeric value of the expression `ppa`?

5) **[1 Mark]** What is the numeric value of the expression `*ppa + 2`?

## Part II: Practical Programming

This part will test whether you can apply the conceptual knowledge you have learned in Weeks 3–5 to solve practical programming tasks. You may only use the Standard C Library to perform the tasks in this part.

In the programming tasks, you will implement a simple database management system (DBMS). DBMS is a systems program that performs storage, retrieval, and updating of data in a computer system. DBMS addresses two major problems in conventional file-based systems: data redundancy and data dependence.

**Sample code showing an example on how you can test your code are provided under the `files` directory in the archive that contains this file.**

### Commenting

You should provide appropriate comments to make your source code readable. If your code does not work and there are no comments, you may lose all marks.

### Coding Style

You should follow a consistent coding style when writing your source code. Coding style (aka coding standard) refers to the use of appropriate indentation, proper placement of braces, proper formatting of control constructs, and many others. Following a particular coding style consistently will make your source code more readable.

There are many coding standards available (search "C coding style"), but we suggest you consult the *lightweight* Linux kernel coding style (see `https://www.kernel.org/doc/html/v4.10/process/coding-style.html`). The relevant sections are Sections 1, 2, 3, 4, 6, and 8. Note that you do not have to follow every recommendation you can find in a coding style document, you just have to apply that style consistently.

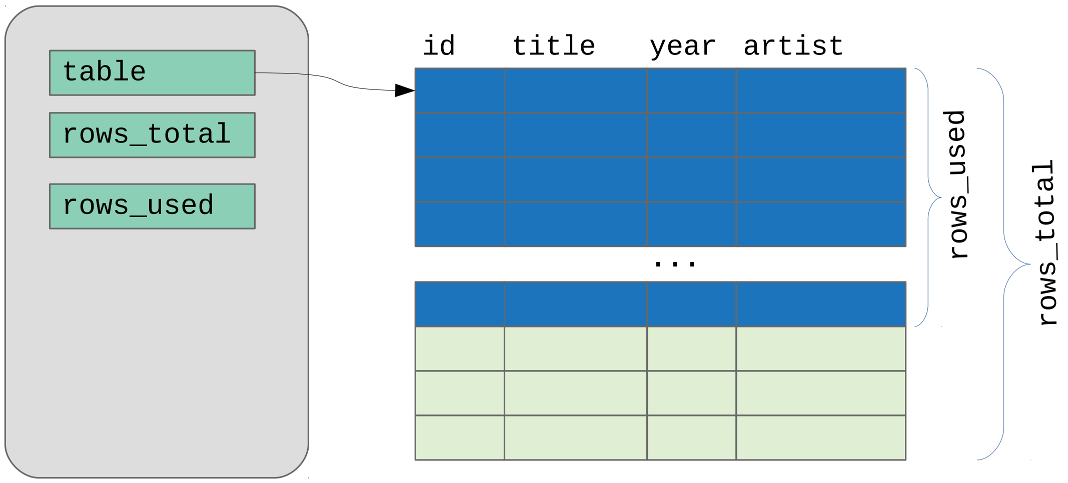
**Program Design**

A fundamental concept in DBMS is the *table*. A table consists of zero or more *records* or entries, and each record can have one or more *fields* or columns. An example of a table that stores information about music albums is shown below:

| id | title | year | artist |
|---|---|---|---|
| 10 | The Dark Side of the Moon | 1973 | Pink Floyd |
| 14 | Back in Black | 1980 | AC/DC |
| 23 | Their Greatest Hits | 1976 | Eagles |
| 37 | Falling into You | 1996 | Celine Dion |
| 43 | Come Away With Me | 2002 | Norah Jones |
| 55 | 21 | 2011 | Adele |

This table contains 6 records. Each record has 4 fields, namely, `id`, `title`, `year`, and `artist`.

**In this assignment, you will focus on implementing a single database table with 4 fields (id, title, year, and artist).** To guide you in the implementation, a high-level design of the table is shown below:



`db_table` Structure Array of `album` Structures

The design consists of a `db_table` structure and an array of `album` structures. The `db_table` structure should have the following member variables:

- `table`: a pointer to an array of structures that is dynamically allocated,

- `rows_total`: the total number of rows in the array of structures,
- `rows_used`: the number of rows in the array of structures that contains valid records.

The array of `album` structures will hold the records. This array should be dynamically allocated and adjusted using the following scheme:

- Initially, the array should be empty. `rows_total` should be set to 0 and `rows_used` should be set to 0.
- When adding a record and there is unused space in the table (`rows_used` is less than `rows_total`), the record is stored in the next unused row, and `rows_used` is updated accordingly.
- When adding a record and there is no space in the table (`rows_used` is equal to `rows_total`), additional memory for holding 5 records should be allocated. Then, the record is stored in the next unused row, and `rows_used` and `rows_total` are updated accordingly.
- When removing a record, records after the removed record should be moved up by one row so that there will be no gaps in between used rows. When the number of unused rows reaches 5, the unused rows should be released, and `rows_used` and `rows_total` are updated accordingly.

The definition of the `db_table` and `album` structures are given in the header file named `dbms.h` which is provided under the `files` directory.

## Task 4.

**Core [45 Marks]**

Provide an implementation of a function called `db_show_row` for displaying the information stored in a row. You are free to format the print out, but all fields of the row should be displayed. The function prototype is given in the header file named `dbms.h` which is provided under the `files` directory.

The function accepts two input parameters:

- Pointer to a `db_table` structure (see definition in `dbms.h`.)
- Unsigned integer indicating the row number of the record to be displayed.

If the record exists, the function should return `1`, otherwise, it should return `0`.

Save the function implementation in a C source file named `dbms.c`.

You can test your function implementation by compiling `dbms.c` together with the file named `t4test.c` which is provided under the `files` directory. Read the contents of `t4test.c` to know how to compile and run the resulting executable file.

## Task 5.

**Completion [12 Marks]**

Provide an implementation of a function called `db_add_row` for a adding a record into the database table. The function prototype is given in the header file named `dbms.h` which is provided under the `files` directory.

The function accepts two input parameters:

- Pointer to a `db_table` structure (see definition in `dbms.h`.)
- Pointer to `album` structure (see definition in `dbms.h`) as input parameter containing the record details to be stored in the table.

As mentioned in the *Program Design* section:

- When adding a record and there is unused space in the table (`rows_used` is less than `rows_total`), the record is stored in the next unused row, and `rows_used` is updated accordingly.
- When adding a record and there is no space in the table (`rows_used` is equal to `rows_total`), additional memory for holding 5 records should be allocated. Then, the record is stored in the next unused row, and `rows_used` and `rows_total` are updated accordingly.

The function should always return `1` unless the allocation of additional memory (when needed) fails where it should return `0`.

Save the function implementation in a C source file named `dbms.c`.

You can test your function implementation by compiling `dbms.c` together with the file named `t5test.c` which is provided under the `files` directory. Read the contents of `t5test.c` to know how to compile and run the resulting executable file.

**Task 6.**

**Challenge [10 Marks]**

Provide an implementation of a function called `db_remove_row` for a removing a record from the database table. The function prototype is given in the header file named `dbms.h` which is provided under the `files` directory.

The function accepts two input parameters:

- Pointer to a `db_table` structure (see definition in `dbms.h`.)
- Unsigned long integer specifying the id of the record to be removed.

As mentioned in the *Program Design* section:

- When removing a record, records after the removed record should be moved up by one row so that there will be no gaps in between used rows. When the number of unused rows reaches 5, the unused rows should be released, and `rows_used` and `rows_total` are updated accordingly.

The function should return `1` if the removal was successful, otherwise, it should return `0`.

Save the function implementation in a C source file named `dbms.c`.

You can test your function implementation by compiling `dbms.c` together with the file named `t6test.c` which is provided under the `files` directory. Read the contents of `t6test.c` to know how to compile and run the resulting executable file.

**Marking Criteria for Tasks 4–6:**

| **Criteria** | **Weight** | **Expectations for Full Marks** |
|---|---|---|
| Compilation | 10% | Compiles without warnings |
| Commenting | 10% | Sufficient and appropriate comments |
| Coding Style | 10% | Consistent coding style |
| Memory Allocation | 30% | Uses dynamic memory allocation functions correctly |
| Correctness | 40% | Handles all test cases correctly |
| Total | 100% | |