# CSA2001
# Programming Paradigms Assignment 1
# Imperative Paradigm

Mark Galea
mgale11@um.edu.mt

17th October 2016

*The aim of this assignment is to strengthen your understanding of the imperative programming paradigm. For the first task you will build a Turing Machine which executes simple arithmetic addition while for the second task you will implement an API for Set operations. Lastly you will write a short report.*

## 1 Preliminaries

This is the first assignment for CSA2001; three more will follow with time. It carries 10% of the final mark and hence will be scored out of 10 points as follows:

- **4 points** for Task 1
- **5 points** for Task 2
- **1 point** for clear and detailed report

Parts of the assignment may also be present in the exam. The deadline for submission of the complete assignment is at **NOON** on **31st of October 2016** and it must be carried out **individually**.

You are reminded that we take plagiarism seriously. While you are encouraged to discuss ideas with fellow students and perform additional research you cannot copy and steal ideas. Read and follow the information found at `https://www.um.edu.mt/ict/Plagiarism` carefully. Extreme measures might be taken in case of plagiarism.

## 2 Task 1

You are to program a Turing machine that performs arithmetic addition of two integers such as $1 + 2$).

Draw up the State-Transition Diagram for such a Turing Machine showing clearly the start and **halting** states and all the **transition labels**. Describe clearly the **representation** that you have decided to adopt and the **alphabet** being used.

Provide a verbose dry-run of:

1. $(1 + 5)$
2. $(4 + 2)$

At each step clearly list transition details including the current state and the resultant state.

*Remember that in this exercise efficiency is not important but simplification is! Make sure you do not lose information when simplifying representation of data.*

## 3 Task 2

You are to implement a set of functions providing various Set operations. This task must be carried out in **C**.

Implement the functions corresponding to the following header file **sets.h**

```c
#include <stdlib.h>
#include <stdio.h>

#define TRUE 1
#define FALSE 0
typedef int (*filter_fn)(int);
typedef int (*map_fn)(int);

typedef struct set_element {
   int value;
   struct set_element* next;
} set_element;

set_element* create(int elem);
void destroy(set_element* head);
int add(set_element* src, int elem);
int delete(set_element* src, int elem);
```

```
int contains(set_element* src, int elem);
set_element* set_union(set_element* first, set_element* second);
set_element* set_diff(set_element* first, set_element* second);
set_element* set_intersection(set_element* first, set_element*
    second);
void print_set(set_element* start);
int for_all(set_element* set, filter_fn filter);
int exists(set_element* set, filter_fn filter);
set_element* map(set_element* head, map_fn map);
```

### 3.1 Functions Explanation

| Name | Definition |
|---|---|
| `create` | Initiates the set structure and adds the first element. |
| `destroy` | Deletes the set and all of its elements. |
| `add` | Add an element to the set. |
| `delete` | Removes an element from the set. |
| `contains` | Checks whether an element is contained within a set. |
| `set_union` | Returns the union of two sets. |
| `set_diff` | Returns the difference between two sets. |
| `set_intersection` | Returns the intersection of two sets. |
| `print_set` | Prints a set in comma-separated fashion. |
| `for_all` | Checks whether the given filter function is satisfied by all the elements of the set.* |
| `exists` | Checks whether there exists at least one element in the set that satisfies the given filter function.* |
| `map` | Maps the elements of one set to another. |

### 3.2 For all and Exists*

Here is an example of what these two functions mean: Consider set $\mathcal{S}_1 = \{1, 2, 3\}$ and the `is_even` filter function which returns true only if the element of a set is even.

Calling the `for_all` function for $\mathcal{S}_1$ and `is_even`, this will return `false` since 1 and 3 are not even. On the other hand calling the `exists` function for $\mathcal{S}_1$ and `is_even`, this will return `true` since the set contains 2 which is even.

### 3.3 Notes

This file containing the implemented functions should be called **sets.c**. Another file, named **main.c**, should contain the main function which performs set operations of your choice by calling the API defined in **sets.c**. (Only the implementation of the set operations i.e **sets.c** is assessed)

1. Do not use any software packages or libraries that provide out-of-the-box linked lists or set operations implementations.
2. Do not use network services such as online APIs.
3. Comment your code when necessary but do not comment the obvious.
4. Test your code.
5. Keep memory allocation (`malloc` & `free`) in mind. Your programs will be tested for *memory leaks*.

## 4 Report

This assignment is to be accompanied by a concise report outlining the remarks, decisions or assumptions you have made while carrying out the two tasks. The full working for Task 1 is to be provided in this report.

You are allowed a **maximum of 3 pages** for the report; shorter reports are welcome.

## 5 Submissions

All submissions have to take place by the stipulated deadline.

1. Submit a hard copy of the report to Ms. Vanessa Borg.
2. Package all the source code and the report and email them to me on my email address. The package should hence include the **report.pdf** and a **src** folder with **sets.h**, **sets.c** and **main.c**.
   (a) Make sure that the files are named exactly as specified in this document.
   (b) Identify yourself with student code / ID card number in the email.

## 6 Conclusion

If you have reasonable questions please contact me on my email address. Obviously I will not provide implementation details however if you require some clarification feel free to contact me. Dedicate enough time for completing this assignment since no extensions will be provided. Good luck!