# MECHTRON 2MP3 : FINAL EXAMINATION

Fall 2022

Instructor: Nicholas Moore

**Exam Start Time: 9:00 AM, Monday Dec. 20th, 2022**

**Exam End Time: 11:30 AM, Monday Dec. 20th, 2022**

Maximum Grade: 32/32

## Instructions

THIS EXAMINATION PAPER INCLUDES 5 QUESTIONS ON 4 PAGES. YOU ARE RESPONSIBLE FOR ENSURING YOUR COPY OF THE PAPER IS COMPLETE. BRING ANY DISCREPANCIES TO THE ATTENTION OF THE COURSE INSTRUCTOR.

**Permitted Materials**:

(cell phones are not permitted)

- Laptop Computer
- McMaster Standard Calculator (CASIO FX-991MS/MS+)
- One Double Sided 8.5 x 11 Crib Sheet

## Special Instructions - Please Read

- Do not rename the files or folders contained in the "Exam.zip" compressed archive.
- Do not use function names other than those provided.
- Do not change the arguments of functions. Use them as provided or indicated in the question.
- You must submit your exam using your MT2MP3 private github repository. Your final commit must be timestamped prior to the end time of the exam.
- There is no grace period for exam submissions.
- Solutions submitted with syntax errors are worth zero points.
- In this exam, the length of the instructor's solutions to these questions has been provided to help you guage the length and difficulty of the problems. You are not required to find solutions that are close to or under the lines of code indicated, there are no grade deductions for solutions that are longer than required. These line of code counts include function declaration lines and lines containing closing curly braces.

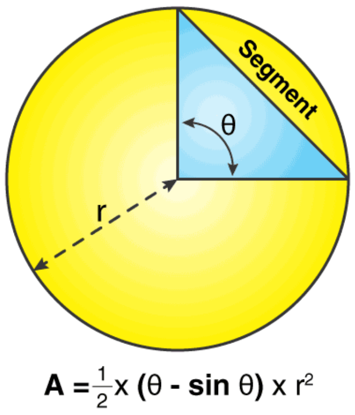
## Submission Instructions

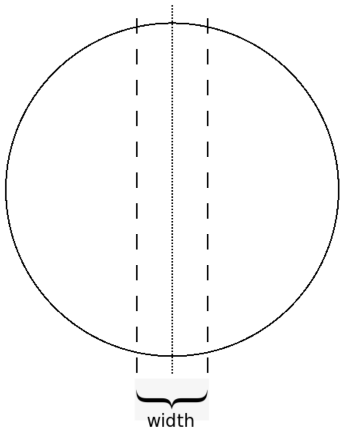
- Create a new folder called `E` in your `MT2MP3` submission repository on the department's gitlab server.
- Code templates are available as a zip file from Avenue. You may simply decompress the archive inside `MT2MP3/E`, and the correct directory structure and file names will be added automatically.
- Be sure to `add`, `commit` and `push` your answers *before* the exam stop time. You can verify your submission via web browser by inspecting the contents of your submission repo.

Good luck!

## Questions

1. (5 points) **A Piece of Cake**
There are better and worse ways to cut a piece of cake. The traditional, wedge-based approach is generally inferior, as it exposes the open surfaces of the cake to air, thus causing staleness. A better method is to take a section out of the middle and push the two halves together, thus blocking access. The formula for the area of a segment of a circle is given in the diagram below, as well as what is meant by the "width" of a slice.





Some hints for this question:

- The value of $\pi$ is defined as `M_PI` in `math.h`.
- Inverse triggonometric functions are available in the math library as `acos`, `asin` and `atan` for "arc-cos" etc.
- The triggonometric functions in the math library work in radians, not degrees.
- Don't forget to add `-lm` to your gcc invokation to link the math library!

(a) (1 point) Write a function, `area`, which, given the radius of a circle, returns its area.

(b) (3 points) Write a function `pieceOfCake`, which, given the radius of a circle and the width of a cut of the cake as given above, calculates how much area is in the segments left behind by the cut. The cut will always be centered exactly over the centreline of the circle. In other words, the two segments left over after the cut will always be a perfect mirror image of each other.

(c) (1 point) Write a function `pieceOfCakeComplement`, which, given the same inputs as 1b, returns the area of the piece of cake that has been cut.

2. (4 points) **Van Eck's Sequence**
Van Eck's Sequence is defined as follows:

- The sequence always begins with zero.
- For the $(n+1)^{th}$ number, consider the $n^{th}$ number.
  - Step backwards, one number at a time, through the sequence.
  - The $(n+1)^{th}$ is the number of times you had to step backwards to find the value of the $n^{th}$ occur again in the sequence.
    - In other words, the $(n+1)^{th}$ number is the number of numbers it's been, relative to the $n^{th}$ number, since the $n^{th}$ number last occured in the sequence.
  - If the $n^{th}$ number has never before occured in the sequence, the $(n+1)^{th}$ number is zero.

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Van Eck | 0 | 0 | 1 | 0 | 2 | 0 | 2 | 2 | 1 | 6 | 0 | 5 | 0 |

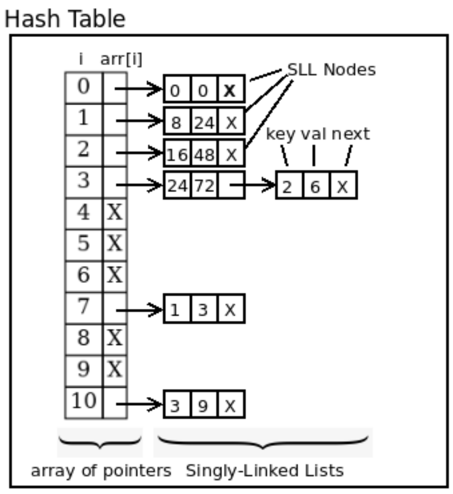
Write a C function, `vaneck`, which, given the size of a Van Eck's sequence to calculate, returns a pointer to that sequence.

- HINT: what do we do to stop an array being deallocated when the function that declared it terminates? The answer is not make it global!

3. (13 points) **Don't Make a Hash of Things!**
A hash table is a highly efficient data structure for searching and retrieving key/value pairs. You have been provided the following structure definitions in `Q1/hashtable.c`

```
struct sll {
    int key;
    int val;
    struct sll* next;
};
```

```
struct hTable {
        int size;
    struct sll** arr;
};
```



The hash table itself is an array of pointers to singly-linked lists. The way it works, is that keys are transformed via a "hash function" into "hash values". The hash value is used as the index at which the key-value pair is stored in an array (or "slot").

The complication is that multiple keys can hash to the same slot, so we can't store the key-value pairs directly in the array. Instead, each slot points to a singly linked list containing all the key-value pairs which have been added to the table and which hash to that slot.

(a) (3 points) `initTable`

This function returns a pointer to a newly created hash table, given the `size` of the table as input. The newly created hash table should contain both the `size` that was given, and a pointer to an array of singly linked list pointers called `arr`. The size of this array is, naturally, `size`, as input to the function. *(Instructor's solution: 9 lines of code)*

(b) (1 point) `hash`

This function takes a `key` and the `size` of the hash table (both integers). The return result of this function is a lucky number (777) multiplied by the key, modulus the size of the table, so that all keys hash to a slot that actually exists in the table. *(Instructor's solution: 3 lines of code)*

(c) (3 points) `insert`

The hash function inserts a new key-value pair into the hash table. The return result of the `hash` function, when called on a particular key, is the index in the array of pointers where the new node is to be stored. New nodes should be *prepended* to the linked list pointed to by the correct entry in the array. *(Instructor's solution: 8 lines of code, but I used a ternary expression, uses* `hash`*)*

(d) (3 points) `walkSLL`
Basic linear search of a linked list. Given a pointer to the first node in a linked list, `walkSLL` walks the data structure until the key of the current node matches the key we are looking for. At this point, we return the *value* associated with that node. If the end of the list is reached and we haven't had a matching key, return -1 to indicate failure. *(Instructor's solution: 11 lines of code, recursive)*

(e) (3 points) `lookup`
The lookup function takes a hash table and a key to search, and tries to return the value associated with that key in the data structure. Naturally, in order to find it, we have to first find the hash value of the key, and then walk the linked list pointed to by the corresponding array entry. As with `walkSLL`, we return the value associated with the key if it's in the table, and -1 if it isn't. *(Instructor's solution : 8 lines of code, uses `hash` and `walkSLL`)*

You have been provided two functions, `printSLL` and `printTable` to help get you started, as well as some test cases.

4. (6 points) **The People's Sorting Algorithm**
`stalinSort` is a destructive sorting algorithm where each element of the array you are sorting is examined in turn. Any elements not conforming to the property of sortedness are labelled enemies of the people, dangerous reactionaries, counter-revolutionaries or some such term, and are "re-educated" (that is to say, deleted... permanently...).

Another way to say this is that the algorithm removes any element that is smaller than the maximum element in the sequence so far.

This function takes an array to be sorted and the size of the array as arguments, and outputs a pointer to an array containing the sorted elements. Note that the array of sorted elements may (and probably will) be smaller than the array you started with.

*(Instructor's solution: 26 lines of code)*

5. (4 points) **Take a Byte out of Crime!**
Write a function, `incBytes` that takes three arguments:

- `void* array` ⇒ A pointer to the array we wish to process
- `int size_of_type` ⇒ The size (in bytes) of the elements of the array we are processing.
- `int size_of_array` ⇒ The size (in elements) of the array we are processing.

And increments each byte in the specified region of memory by 1. Note: your result must respect overflow rules. That is, if this addition would overflow on any byte, your algorithm should prevent it. For example, incrementing the two adjacent bytes 0x00FF would result in 0x0100, not 0x0200. 0x00FF + 0x0001 = 0x0100, but this overflow bit is ignored in this algorithm.

*(Instructor's solution: 3 lines of code, 5 for readability. This question can be a one-liner, but finding a one-line solution is neither required nor recommended.)*

– END OF EXAM –