

CS 496: Assignment 1

Due: 4 February, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This project seeks to help you obtain some practice with lists and tuples. A Mini-Logo program is just a list of instructions (integers). The encoding of Mini-Logo instructions is as follows:

Encoding	Instruction
0	Pen down
1	Pen up
2	Move North
3	Move East
4	Move South
5	Move West

An example of a program that draws a square:

```
[0; 2; 3; 4; 5; 1]
```

This other one draws two concentric squares

```
[0;2; 2; 2; 2; 3; 3; 3; 4; 4; 4; 4; 5; 5; 5; 5; 2; 2; 2; 2; 2; 2; 2; 2; 3; 3; 3; 3; 3; 3;
  ↪ 3; 3; 4; 4; 4; 4; 4; 4; 4; 4; 5; 5; 5; 5; 5; 5; 5; 5; 1]
```

This last one draws the letter 'E':

```
[0;2;2;3;3;5;5;4;3;5;4;3;3;5;5;1]
```

2.1 Encoding Mini-Logo Programs

We will use the following user defined datatypes.

```
1 type program = int list
```

For example:

```
1 let square : program = [0; 2; 2; 3; 3; 4; 4; 5; 5; 1]
2 let letter_e : program = [0;2;2;3;3;5;5;4;3;5;4;3;3;5;5;1]
```

declares `square` to be a program.

2.2 Exercises

We illustrate some of the functions below assuming we have the declaration:

```
1 let letter_e = [0;2;2;3;3;5;5;4;3;5;4;3;3;5;5;1]
```

Note: you may not change the names of the functions you are asked to implement below, nor the number of arguments they take, nor the types of their arguments.

1. Implement a function

```
mirror_image : int list -> int list
```

that returns a program that draws the mirror image of the input program. For example

```
1 # mirror_image letter_e;;
2 - : int list = [0; 4; 4; 5; 5; 3; 3; 2; 5; 3; 2; 5; 5; 3; 3; 1]
```

Hint: use `map`.

2. Implement a function

```
rotate_90_letter : int list -> int list
```

that given a program returns a new one which draws the same pictures except that they are rotated 90 degrees. For example:

```
1 # rotate_90_letter letter_e;;
2 - : int list = [0; 3; 3; 4; 4; 2; 2; 5; 4; 2; 5; 4; 4; 2; 2; 1]
```

Hint: use `map`.

3. Implement a function

```
rotate_90_word : int list list -> int list list
```

that given a list of programs that represent letters returns a new list in which each program draws the same pictures except that they are rotated 90 degrees. For example:

```
1 # rotate_90_word [letter_e;letter_e];;
2 - : int list list =
3 [[0; 3; 3; 4; 4; 2; 2; 5; 4; 2; 5; 4; 4; 2; 2; 1];
4 [0; 3; 3; 4; 4; 2; 2; 5; 4; 2; 5; 4; 4; 2; 2; 1]]
```

Hint: use map.

4. Implement a function

```
repeat : int -> 'a -> 'a list
```

such that `repeat n x` returns a list with `n` copies of `x`. For example:

```
1 # repeat 3 "hello" ;;
2 - : string list = ["hello"; "hello"; "hello"]
```

5. Implement a function

```
pantograph : int -> int list -> int list
```

such that `pantograph n p` returns a program that draws the same things as `p` only enlarged `n`-fold. For example:

```
1 # pantograph 2 letter_e;;
2 - : int list =
3 [0; 2; 2; 2; 2; 3; 3; 3; 3; 5; 5; 5; 5; 4; 4; 3; 3; 5; 5; 4; 4; 3; 3;
4 3; 3; 5; 5; 5; 5; 1]
```

Your solution must use map. Implement also a solution `pantograph_nm` without using map. Finally, implement a solution `pantograph_f` using fold.

6. Implement a function

```
coverage : int*int -> int list -> (int*int) list
```

that given a starting coordinate and a program returns the list of coordinates that the program visits. You may introduce helper functions to make your code more readable. Also, you need not concern yourself with repetitions. For example:

```
1 # coverage (0,0) letter_e;;
2 - : (int * int) list =
3 [(0, 0); (0, 0); (0, 1); (0, 2); (1, 2); (2, 2); (1, 2); (0, 2); (0, 1);
4  (1, 1); (0, 1); (0, 0); (1, 0); (2, 0); (1, 0); (0, 0); (0, 0)]
```

7. Implement a function

```
compress : int list -> (int*int) list
```

that compresses a program by replacing adjacent copies of the same instruction with a tuple `(m,n)` where `m` is the instruction and `n` is the number of consecutive times it should be executed. For example,


```
1 # compress letter_e;;
2 - : (int * int) list =
3 [(0, 1); (2, 2); (3, 2); (5, 2); (4, 1); (3, 1); (5, 1); (4, 1); (3, 2)
  ↪ ;(5, 2); (1, 1)]
```

8. Implement a function

```
uncompress : (int*int) list -> int list
```

that decompresses a compressed program. For example,

```
1 # uncompress (compress letter_e);;
2 - : int list = [0; 2; 2; 3; 3; 5; 5; 4; 3; 5; 4; 3; 3; 5; 5; 1]
```

Implement a second solution using map `uncompress_m` and third one using fold `uncompress_f` .

9. Implement a function

```
optimize: program -> program
```

that optimizes a program by eliminating redundant pen up and pen down instructions. For this exercise, you must assume that the pen is initially in the up position. For example,

```
1 # optimize [1];;
2 - : int list = []
3 # optimize [1;1;1;1];;
4 - : int list = []
5 # optimize [1;1;1;1;0];;
6 - : int list = [0]
7 # optimize [1;1;1;1;0;1;0;1];;
8 - : int list = [0; 1; 0; 1]
9 # optimize [1;1;1;1;0;1;0;1;1;1;1];;
10 - : int list = [0; 1; 0; 1]
11 # optimize [0;1;0;1];;
12 - : int list = [0; 1; 0; 1]
13 # optimize [2;3;4;5];;
14 - : int list = [2; 3; 4; 5]
```

Hint: use a helper function that has an additional argument that carries the current state of the pen.

3 Submission instructions

Submit a single file named `hw1.ml` through Canvas. No report is required. Your grade will be determined as follows:

- You will get 0 points if your code does not compile.
- Partial credit may be given for style, comments and readability.