Appendix: Function Definitions

This appendix contains the function definitions (arguments, return values, error codes) for all the provided utility functions and all the functions you need to implement.

Utilities: Memory Allocation

`malloc`

`malloc`: Allocates heap memory.			
Arguments	`a0`	`int`	The size of the memory that we want to allocate (in bytes).
Return values	`a0`	`void *`	A pointer to the allocated memory. If the allocation failed, this value is 0.

`free`

`free`: Frees heap memory.			
Arguments	`a0`	`void *`	A pointer to the allocated memory to be freed.
Return values	None		

Utilities: File Operations

`fopen`

`fopen`: Open a file for reading or writing.			
Arguments	`a0`	`char *`	A pointer to the filename string.
	`a1`	`int`	Permission bits. 0 for read-only, 1 for write-only.
Return values	`a0`	`int`	A file descriptor. This integer can be used in other file operation functions to refer to the opened file. If opening the file failed, this value is -1.

fread

`fread`: Read bytes from a file to a buffer in memory. Subsequent reads will read from later parts of the file.			
Arguments	`a0`	`int`	The file descriptor of the file we want to read from, previously returned by `fopen`.
	`a1`	`int*`	A pointer to the buffer where the read bytes will be stored. The buffer should have been previously allocated with `malloc`.
	`a2`	`int`	The number of bytes to read from the file.
Return values	`a0`	`int`	The number of bytes actually read from the file. If this differs from the argument provided in `a2`, then we either hit the end of the file or there was an error.

fwrite

`fwrite`: Write bytes from a buffer in memory to a file. Subsequent writes append to the end of the existing file.			
Arguments	`a0`	`int`	The file descriptor of the file we want to write to, previously returned by `fopen`.
	`a1`	`void *`	A pointer to a buffer containing what we want to write to the file.
	`a2`	`int`	The number of elements to write to the file.
	`a3`	`int`	The size of each element. In total, `a2` × `a3` bytes are written.
Return values	`a0`	`int`	The number of items actually written to the file. If this differs from the number of items specified (`a2`), then we either hit the end of the file or there was an error.

fclose

`fclose`: Close a file, saving any writes we have made to the file.			
Arguments	`a0`	`int`	The file descriptor of the file we want to close, previously returned by `fopen`.

Return values	`a0`	`int`	0 on success, and -1 otherwise.

Utilities: Printing

`print_int`

`print_int`: Prints an integer.			
Arguments	`a0`	`int`	The integer to print.
Return values	None		

`print_char`

`print_char`: Prints a character.			
Arguments	`a0`	`char`	The character to print. You can provide the ASCII code or put the character directly in the register like `li t0 '\n'`.
Return values	None		

Part A

`relu`

`relu`: Task 2.			
Arguments	`a0`	`int *`	A pointer to the start of the integer array.
	`a1`	`int`	The number of integers in the array. You can assume that this argument matches the actual length of the integer array.
Return values	None		

`argmax`

`argmax`: Task 3.			
Arguments	`a0`	`int *`	A pointer to the start of the integer array.
	`a1`	`int`	The number of integers in the array. You can assume that this argument matches the actual length of the integer array.
Return values	`a0`	`int`	The index of the largest element. If the largest element appears multiple times, return the smallest index.

`dot`

`dot`: Task 4.			
Arguments	`a0`	`int *`	A pointer to the start of the first array.
	`a1`	`int *`	A pointer to the start of the second array.
	`a2`	`int`	The number of elements to use in the calculation.
	`a3`	`int`	The stride of the first array.
	`a4`	`int`	The stride of the second array.
Return values	`a0`	`int`	The dot product of the two arrays, using the given number of elements and the given strides.

`matmul`

`matmul`: Task 5.			
Arguments	`a0`	`int *`	A pointer to the start of the first matrix A (stored as an integer array in row-major order).
	`a1`	`int`	The number of rows (height) of the first matrix A.
	`a2`	`int`	The number of columns (width) of the first matrix A.
	`a3`	`int *`	A pointer to the start of the second matrix B (stored as an integer array in row-major order).
	`a4`	`int`	The number of rows (height) of the second matrix B.
	`a5`	`int`	The number of columns (width) of the second matrix B.

	`a6`	`int *`	A pointer to the start of an integer array where the result C should be stored. You can assume this memory has been allocated (but is uninitialized) and has enough space to store C.
Return values	None		

Testing functions

Loss functions: Task 6.			
Arguments	`a0`	`int *`	A pointer to the start of the first input array.
	`a1`	`int *`	A pointer to the start of the second input array.
	`a2`	`int`	The number of integers in the array.
	`a3`	`int *`	A pointer to the start of the output array, where the results will be stored.
Return values	`a0`	`int`	The sum of the elements in the output array. (No return value for zero-one loss.)

`initialize_zero`: Task 6.			
Arguments	`a0`	`int`	The size of the array to be created.
Return values	`a0`	`int *`	A pointer to the newly-allocated array of zeros.

Part B

`read_matrix`

`read_matrix`: Task 7.			
Arguments	`a0`	`char *`	A pointer to the filename string.
	`a1`	`int *`	A pointer to an integer which will contain the number of rows. You can assume this points to allocated memory.

	`a2`	`int *`	A pointer to an integer which will contain the number of columns. You can assume this points to allocated memory.
Return values	`a0`	`int *`	A pointer to the matrix in memory.

`write_matrix`

`write_matrix`: Task 8.			
Arguments	`a0`	`char *`	A pointer to the filename string.
	`a1`	`int *`	A pointer to the matrix in memory (stored as an integer array).
	`a2`	`int`	The number of rows in the matrix.
	`a3`	`int`	The number of columns in the matrix.
Return values	None		

`classify`

`classify`: Task 9.			
Arguments	`a0`	`int`	`argc` (the number of arguments provided)
	`a1`	`char **`	`argv`, a pointer to an array of argument strings (`char *`)
	`a1[1]` = `*(a1 + 4)`	`char *`	A pointer to the filepath string of the first matrix file `m0`.
	`a1[2]` = `*(a1 + 8)`	`char *`	A pointer to the filepath string of the second matrix file `m1`.
	`a1[3]` = `*(a1 + 12)`	`char *`	A pointer to the filepath string of the input matrix file `input`.
	`a1[4]` = `*(a1 + 16)`	`char *`	A pointer to the filepath string of the output file.

	`a2`	`int`	If set to 0, print out the classification. Otherwise, do not print anything.
Return values	`a0`	`int`	The classification (see above).

Error Codes

Part A

Return code	Exception	Functions
26	`malloc` returns an error.	`initialize_zero` (6)
36	The length of the array is less than 1.	`relu` (2), `argmax` (3), `dot` (4), loss functions (6), `initialize_zero` (6)
37	The stride of either array is less than 1.	`dot` (4)
38	The height or width of either matrix is less than 1.	`matmul` (5)
38	The number of columns (width) of the first matrix A is not equal to the number of rows (height) of the second matrix B.	`matmul` (5)

Part B

Return code	Exception	Functions
26	`malloc` returns an error.	`read_matrix` (7), `classify` (9)
27	`fopen` returns an error.	`read_matrix` (7), `write_matrix` (8)
28	`fclose` returns an error.	`read_matrix` (7), `write_matrix` (8)
29	`fread` does not read the correct number of bytes.	`read_matrix` (7)

30	`fwrite` does not write the correct number of bytes.	`write_matrix` (8)
31	There are an incorrect number of command line arguments.	`classify` (9)