Rajzfelismerő

Generated by Doxygen 1.9.1

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 Canvas Struct Reference

`#include <canvas.h>`

Data Fields

- Vector m

3.1.1 Detailed Description

A Canvas struct contains a 2D matrix of 'double' values.

Definition at line 6 of file canvas.h.

3.1.2 Field Documentation

3.1.2.1 m

`Vector m`

This Vector contains the Canvas's width amount of Vectors. These other Vectors represent the rows of the 2D matrix.

Definition at line 11 of file canvas.h.

The documentation for this struct was generated from the following file:

- src/headers/canvas.h

3.2 MLP Struct Reference

`#include <mlp.h>`

Data Fields

- size_t x
- size_t y
- size_t kx
- size_t ky
- char * name
- Vector layers
- Canvas canvas

3.2.1 Detailed Description

Definition at line 14 of file mlp.h.

3.2.2 Field Documentation

3.2.2.1 canvas

`Canvas canvas`

Definition at line 18 of file mlp.h.

3.2.2.2 kx

`size_t kx`

Definition at line 15 of file mlp.h.

3.2.2.3 ky

`size_t ky`

Definition at line 15 of file mlp.h.

3.2.2.4 layers

```
Vector layers
```

Definition at line 17 of file mlp.h.

3.2.2.5 name

```
char* name
```

Definition at line 16 of file mlp.h.

3.2.2.6 x

```
size_t x
```

Definition at line 15 of file mlp.h.

3.2.2.7 y

```
size_t y
```

Definition at line 15 of file mlp.h.

The documentation for this struct was generated from the following file:

- src/headers/mlp.h

3.3 Node Struct Reference

```
#include <mlp.h>
```

Data Fields

- double value
- double bias
- double output
- double(* act)(double)
- Vector con

3.3.1 Detailed Description

Definition at line 6 of file mlp.h.

3.3.2 Field Documentation

3.3.2.1 act

```
double(* act) (double)
```

Pointer to activation function.

Definition at line 9 of file mlp.h.

3.3.2.2 bias

```
double bias
```

Definition at line 7 of file mlp.h.

3.3.2.3 con

```
Vector con
```

Connection weights to each Node in the next layer.

Definition at line 11 of file mlp.h.

3.3.2.4 output

```
double output
```

Definition at line 7 of file mlp.h.

3.3.2.5 value

```
double value
```

Definition at line 7 of file mlp.h.

The documentation for this struct was generated from the following file:

- src/headers/mlp.h

3.4 ReadResult Struct Reference

```
#include <filehandler.h>
```

Data Fields

- RSTATUS status
- MLP model

3.4.1 Detailed Description

Contains a status code and an MLP if the reading was successful.

Definition at line 22 of file filehandler.h.

3.4.2 Field Documentation

3.4.2.1 model

```
MLP model
```

The returned model if the reading was successfuly. Empty otherwise.

Definition at line 26 of file filehandler.h.

3.4.2.2 status

```
RSTATUS status
```

The status code of the reading. Can be SUCCESS or an error code.

Definition at line 24 of file filehandler.h.

The documentation for this struct was generated from the following file:

- src/headers/filehandler.h

3.5 Vector Struct Reference

`#include <vector.h>`

Data Fields

- void * arr
- size_t elem_size
- size_t size
- size_t cap

3.5.1 Detailed Description

Valami leírás

Definition at line 11 of file vector.h.

3.5.2 Field Documentation

3.5.2.1 arr

`void* arr`

asd

Definition at line 13 of file vector.h.

3.5.2.2 cap

`size_t cap`

IDK

Definition at line 16 of file vector.h.

3.5.2.3 elem_size

`size_t elem_size`

Szöveg1

Definition at line 14 of file vector.h.

3.5.2.4 size

`size_t size`

Szöveg2

Definition at line 15 of file vector.h.

The documentation for this struct was generated from the following file:

- src/headers/vector.h

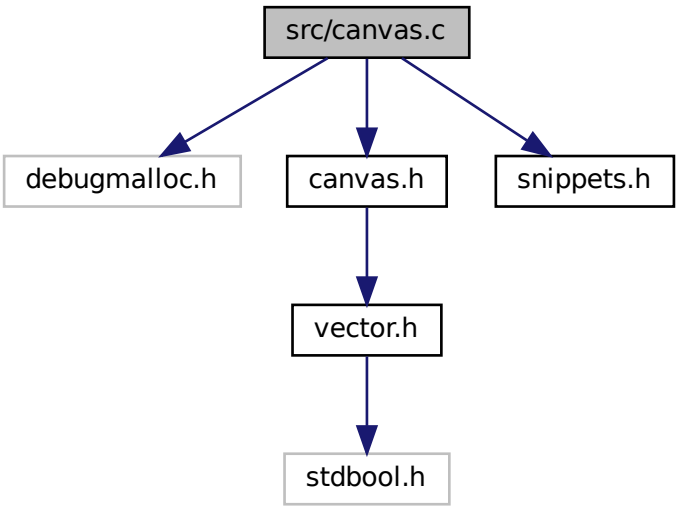
Chapter 4

File Documentation

4.1 src/canvas.c File Reference

```
#include "debugmalloc.h"
#include "canvas.h"
#include "snippets.h"
```

Include dependency graph for canvas.c:



Functions

- Canvas create_canvas (size_t width, size_t height)
- void free_canvas (Canvas *canvas)
- double get_canvas_xy (const Canvas *canvas, size_t x, size_t y)

4.1.1 Function Documentation

4.1.1.1 create_canvas()

```
Canvas create_canvas (
            size_t width,
            size_t height )
```

Creates a new Canvas with the given width and height. The returned Canvas should be freed by the caller, as it contains dynamically allocated memory.

Parameters

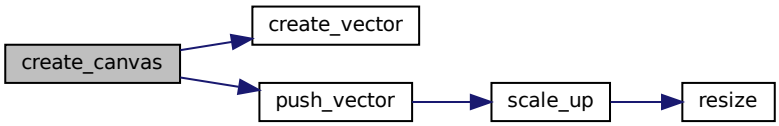
width	The width of the canvas.
height	The height of the canvas.

Returns

The new Canvas struct.

Definition at line 5 of file canvas.c.

Here is the call graph for this function:



4.1.1.2 free_canvas()

```
void free_canvas (
            Canvas * canvas )
```

Frees all dynamically allocated memory used by a Canvas.

Parameters

canvas	Pointer to the canvas that should be freed.

Definition at line 17 of file canvas.c.

Here is the call graph for this function:

4.1.1.3 get_canvas_xy()

```
double get_canvas_xy (
            const Canvas * canvas,
            size_t x,
            size_t y )
```

Queries the canvas at given coordinates. The (0, 0) coordinate is at the top left corner.

Parameters

canvas	Pointer to the target canvas.
x	The X coordinate on the canvas.
y	The Y coordinate on the canvas.

Returns

The greyscale value on the canvas at the given coordinates.

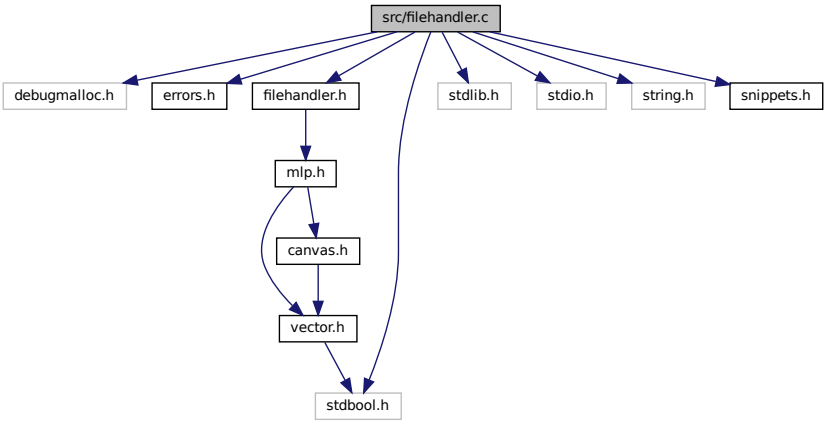
Definition at line 28 of file canvas.c.

4.2 src/filehandler.c File Reference

```
#include "debugmalloc.h"
#include "errors.h"
#include "filehandler.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "snippets.h"
```

```
#include <stdbool.h>
```

Include dependency graph for filehandler.c:



Macros

- #define pass(x, y, f, s) if((x) != (y)) {fclose(f); return (ReadResult){(s), {0}};}
- #define pass(x, y, f, s) if((x) != (y)) {fclose(f); free_mlp(&mlp); return (ReadResult){(s), {0}};}

Functions

- static char * getfilename (const char *path)
- static RSTATUS read_instructions (FILE *f, MLP *mlp, int n)
- static RSTATUS read_biases (FILE *f, MLP *mlp)
- static RSTATUS read_weights (FILE *f, MLP *mlp)
- ReadResult read_model (const char *path)

4.2.1 Macro Definition Documentation

4.2.1.1 pass [1/2]

```
#define pass(
             x,
             y,
             f,
             s ) if((x) != (y)) {fclose(f); return (ReadResult){(s), {0}};}
```

4.2.1.2 pass [2/2]

```
#define pass(
              x,
              y,
              f,
              s ) if((x) != (y)) {fclose(f); free_mlp(&mlp); return (ReadResult){(s), {0}};}
```

4.2.2 Function Documentation

4.2.2.1 getfilename()

```
static char* getfilename (
            const char * path )  [static]
```

Creates a new string with only the file's name and extension extracted from a given file path. The new string should be freed by the calleras it's dynamically allocated.

Parameters

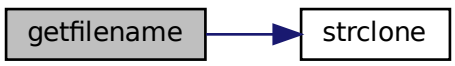
path	The full file path.

Returns

The new string containing the file's name and extension.

Definition at line 19 of file filehandler.c.

Here is the call graph for this function:



4.2.2.2 read_biases()

```
static RSTATUS read_biases (
            FILE * f,
            MLP * mlp )  [static]
```

Reads and processes the biases of each Node in an MLP from a file.

Parameters

f	The file to read from.
mlp	Pointer to the target MLP.

Returns

An RSTATUS with the possible status codes.

Definition at line 87 of file filehandler.c.

Here is the call graph for this function:

4.2.2.3 read_instructions()

```
static RSTATUS read_instructions (
            FILE * f,
            MLP * mlp,
            int n )  [static]
```

Reads and processes a given amount of instructions from a file.

Parameters

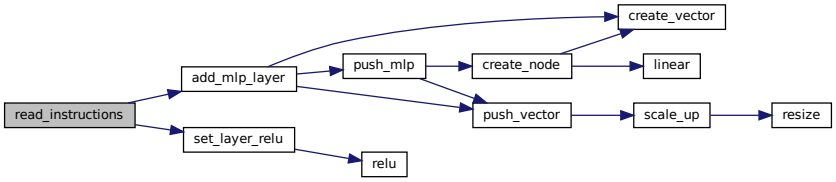
f	The file to read from.
mlp	Pointer to the target MLP.
n	Number of instructions.

Returns

An RSTATUS with the possible status codes.

Definition at line 47 of file filehandler.c.

Here is the call graph for this function:



4.2.2.4 read_model()

```
ReadResult read_model (
            const char * path )
```

Tries to read a model from a file at the given path. The function assumes that the file ends with the .mlpmodel extension.

Parameters

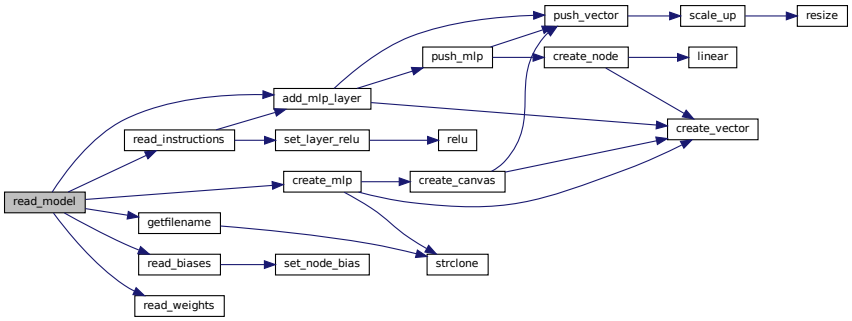
path	The path to the file.

Returns

A ReadResult struct, containing the read's status code and a valid MLP struct if the status code is SUCCESS. If the MLP is valid, then it needs to be freed later by the caller.

Definition at line 136 of file filehandler.c.

Here is the call graph for this function:



4.2.2.5 read_weights()

```
static RSTATUS read_weights (
            FILE * f,
            MLP * mlp )  [static]
```

Reads and processes the necessary weights for each Node in an MLP from a file.

Parameters

f	The file to read from.
mlp	Pointer to the target MLP.

Returns

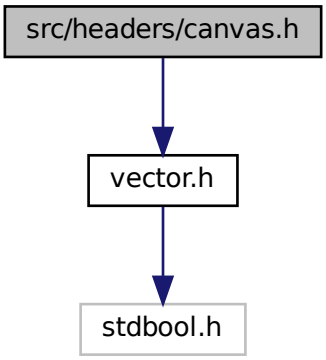
An RSTATUS with the possible status codes.

Definition at line 113 of file filehandler.c.

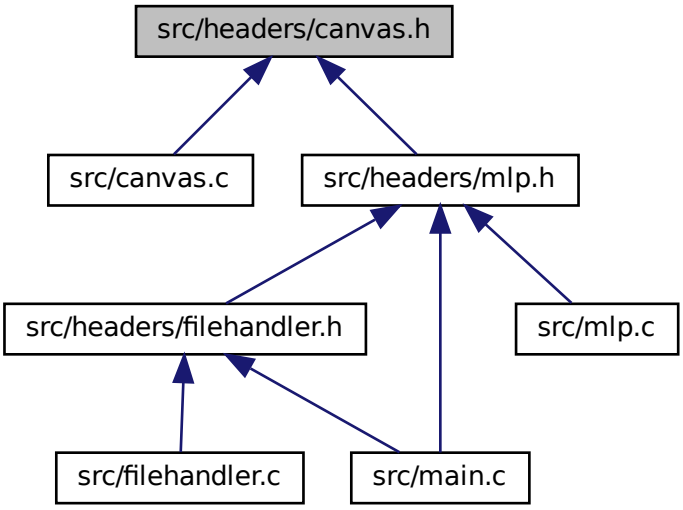
4.3 src/headers/canvas.h File Reference

```
#include "vector.h"
```

Include dependency graph for canvas.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Canvas

Typedefs

- typedef struct Canvas Canvas

Functions

- Canvas create_canvas (size_t width, size_t height)
- void free_canvas (Canvas *canvas)
- double get_canvas_xy (const Canvas *canvas, size_t x, size_t y)

4.3.1 Typedef Documentation

4.3.1.1 Canvas

```
typedef struct Canvas Canvas
```

A Canvas struct contains a 2D matrix of 'double' values.

4.3.2 Function Documentation

4.3.2.1 create_canvas()

```
Canvas create_canvas (
            size_t width,
            size_t height )
```

Creates a new Canvas with the given width and height. The returned Canvas should be freed by the caller, as it contains dynamically allocated memory.

Parameters

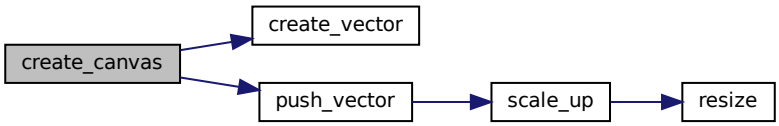
width	The width of the canvas.
height	The height of the canvas.

Returns

The new Canvas struct.

Definition at line 5 of file canvas.c.

Here is the call graph for this function:



4.3.2.2 free_canvas()

```
void free_canvas (
            Canvas * canvas )
```

Frees all dynamically allocated memory used by a Canvas.

Parameters

canvas	Pointer to the canvas that should be freed.

Definition at line 17 of file canvas.c.

Here is the call graph for this function:

4.3.2.3 get_canvas_xy()

```
double get_canvas_xy (
            const Canvas * canvas,
            size_t x,
            size_t y )
```

Queries the canvas at given coordinates. The (0, 0) coordinate is at the top left corner.

Parameters

canvas	Pointer to the target canvas.
x	The X coordinate on the canvas.
y	The Y coordinate on the canvas.

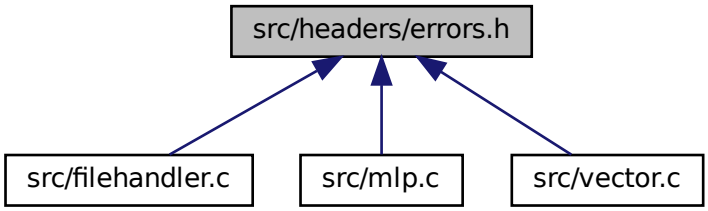
Returns

The greyscale value on the canvas at the given coordinates.

Definition at line 28 of file canvas.c.

4.4 src/headers/errors.h File Reference

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum ERROR ERROR

Enumerations

- enum ERROR { ERR_NULLPOINTER = 1 , ERR_INDEXOUTOFBOUNDS = 2 }

4.4.1 Typedef Documentation

4.4.1.1 ERROR

```
typedef enum ERROR ERROR
```

The program should abort when any of these errors is caught.

4.4.2 Enumeration Type Documentation

4.4.2.1 ERROR

```
enum ERROR
```

The program should abort when any of these errors is caught.

Enumerator

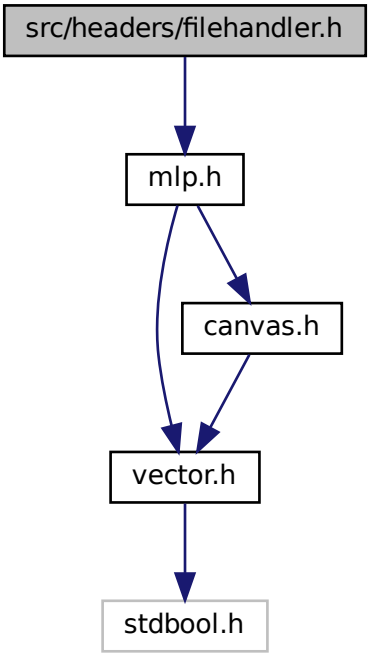
ERR_NULLPOINTER	
ERR_INDEXOUTOFBOUNDS	

Definition at line 6 of file errors.h.

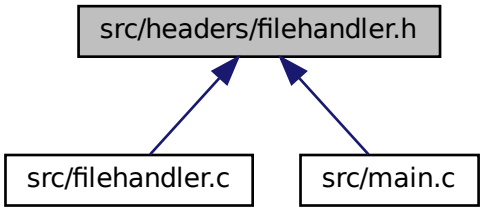
4.5 src/headers/filehandler.h File Reference

```
#include "mlp.h"
```

Include dependency graph for filehandler.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ReadResult

Typedefs

- typedef enum RSTATUS RSTATUS
- typedef struct ReadResult ReadResult

Enumerations

- enum RSTATUS {
  SUCCESS = 0 , NOFILE , NODATA , KERNELSIZE ,
  NOLAYER , WRONGINSTRUCTION }

Functions

- ReadResult read_model (const char ∗path)

4.5.1 Typedef Documentation

4.5.1.1 ReadResult

```
typedef struct ReadResult ReadResult
```

Contains a status code and an MLP if the reading was successful.

4.5.1.2 RSTATUS

```
typedef enum RSTATUS RSTATUS
```

Status codes for the file reading.

4.5.2 Enumeration Type Documentation

4.5.2.1 RSTATUS

```
enum RSTATUS
```

Status codes for the file reading.

Enumerator

SUCCESS	The code finished successfully.
NOFILE	The file couldn't be opened.
NODATA	Couldn't read the requested data from a file or EOF is reached.
KERNELSIZE	The kernel size is invalid for the MaxPool2D operation.
NOLAYER	There isn't enough layers in the MLP.
WRONGINSTRUCTION	The given instruction doesn't exist.

Definition at line 6 of file filehandler.h.

4.5.3 Function Documentation

4.5.3.1 read_model()

```
ReadResult read_model (
            const char * path )
```

Tries to read a model from a file at the given path. The function assumes that the file ends with the .mlpmodel extension.

Parameters

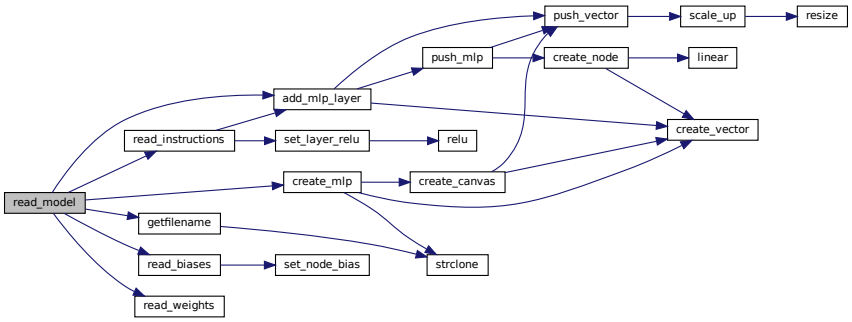
path	The path to the file.

Returns

A ReadResult struct, containing the read's status code and a valid MLP struct if the status code is SUCCESS. If the MLP is valid, then it needs to be freed later by the caller.

Definition at line 136 of file filehandler.c.

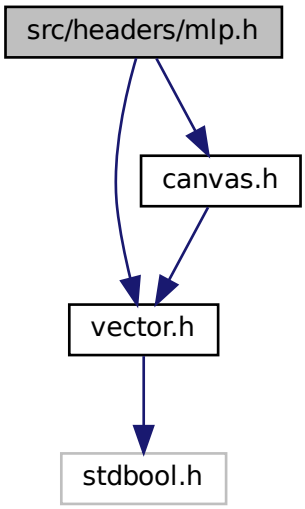
Here is the call graph for this function:



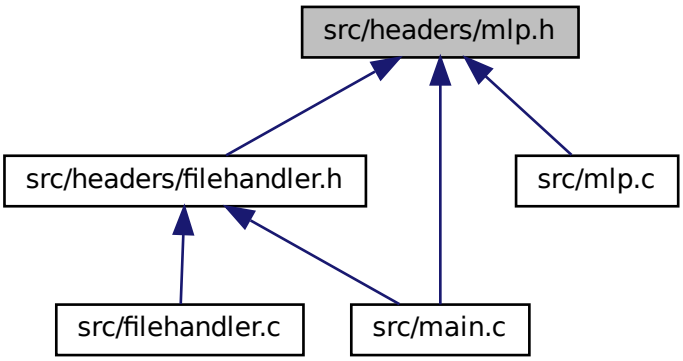
4.6 src/headers/mlp.h File Reference

```
#include "vector.h"
#include "canvas.h"
```

Include dependency graph for mlp.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Node
- struct MLP

Typedefs

- typedef struct Node Node
- typedef struct MLP MLP

Functions

- MLP create_mlp (size_t x, size_t y, size_t kx, size_t ky, char ∗name, size_t layers)
- void add_mlp_layer (MLP ∗mlp, size_t nodes)
- void set_layer_relu (MLP ∗mlp, size_t layer)
- void set_layer_linear (MLP ∗mlp, size_t layer)
- void push_mlp (MLP ∗mlp, size_t layer, double bias)
- void set_node_bias (MLP ∗mlp, size_t layer, size_t n, double bias)
- void free_mlp (MLP ∗mlp)
- void load_mlp_input (MLP ∗mlp)
- void run_mlp (MLP ∗mlp)

4.6.1 Typedef Documentation

4.6.1.1 MLP

```
typedef struct MLP MLP
```

4.6.1.2 Node

```
typedef struct Node Node
```

4.6.2 Function Documentation

4.6.2.1 add_mlp_layer()

```
void add_mlp_layer (
            MLP * mlp,
            size_t nodes )
```

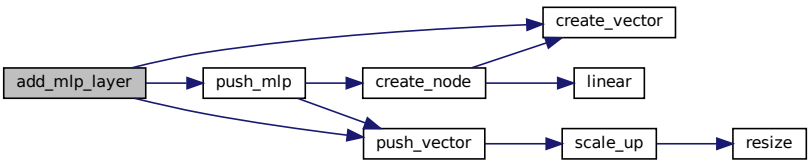
Inserts a new layer at the end of an MLP with a given number of dummy nodes. The new layer will be automatically connected to the layer before it with weights of 1.0 and biases of 0.0.

Parameters

mlp	Pointer to the target MLP.
nodes	Number of dummy nodes.

Definition at line 49 of file mlp.c.

Here is the call graph for this function:



4.6.2.2 create_mlp()

```
MLP create_mlp (
            size_t x,
            size_t y,
            size_t kx,
            size_t ky,
            char * name,
            size_t layers )
```

Creates a new MLP model with a given name and number of layers. The returned MLP should be freed by the caller, as it contains dynamically allocated memory.

Parameters

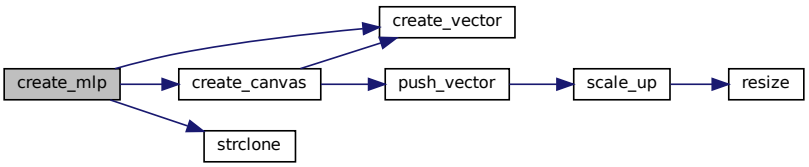
x	Canvas width.
y	Canvas Height.
kx	MaxPool2D kernel width.
ky	MaxPool2D kernel height.
name	String with the name to copy.
layers	Number of layers to start with.

Returns

The newly created MLP struct.

Definition at line 35 of file mlp.c.

Here is the call graph for this function:



4.6.2.3 free_mlp()

```
void free_mlp (
            MLP * mlp )
```

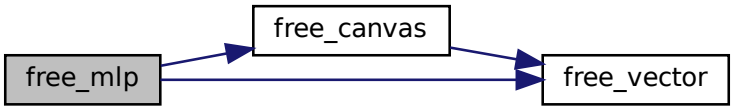
Frees all the dynamically allocated memory used by the model.

Parameters

mlp	Pointer to the MLP.

Definition at line 116 of file mlp.c.

Here is the call graph for this function:



4.6.2.4 load_mlp_input()

```
void load_mlp_input (
            MLP * mlp )
```

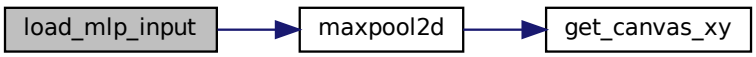
Loads the contents of an MLP's Canvas to the input layer. The MaxPooling is also done by this step.

Parameters

mlp	Pointer to the target MLP.

Definition at line 163 of file mlp.c.

Here is the call graph for this function:



4.6.2.5 push_mlp()

```
void push_mlp (
            MLP * mlp,
            size_t layer,
            double bias )
```

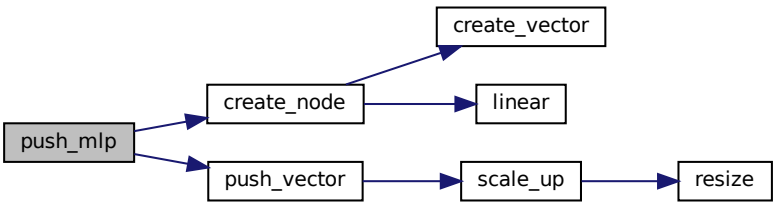
Inserts a new Node at the end of a layer in an MLP. The new Node will have default connections with 1.0 as weight.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.
bias	The new Node's bias.

Definition at line 80 of file mlp.c.

Here is the call graph for this function:



4.6.2.6 run_mlp()

```
void run_mlp (
            MLP * mlp )
```

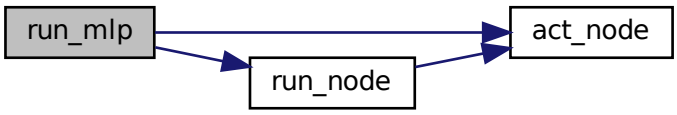
Runs the MLP model in a simple feed-forward manner, calculating the output for the current input layer.

Parameters

mlp	Pointer to the target MLP.

Definition at line 214 of file mlp.c.

Here is the call graph for this function:



4.6.2.7 set_layer_linear()

```
void set_layer_linear (
            MLP * mlp,
            size_t layer )
```

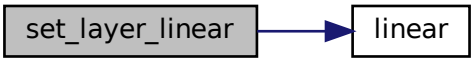
Sets the activation function of all nodes in a layer to linear. The linear function will keep each value as is.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.

Definition at line 69 of file mlp.c.

Here is the call graph for this function:



4.6.2.8 set_layer_relu()

```
void set_layer_relu (
            MLP * mlp,
            size_t layer )
```

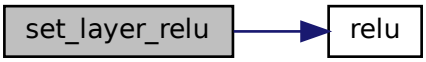
Sets the activation function of all nodes in a layer to ReLU. ReLU will set each negative value to zero and keeps the others.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.

Definition at line 58 of file mlp.c.

Here is the call graph for this function:



4.6.2.9 set_node_bias()

```
void set_node_bias (
            MLP * mlp,
            size_t layer,
            size_t n,
            double bias )
```

Sets a Node's bias in an MLP.

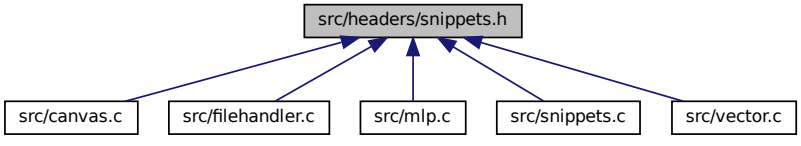
Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.
n	The Node's index insite the layer.
bias	The Node's new bias.

Definition at line 109 of file mlp.c.

4.7 src/headers/snippets.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define max(A, B)
- #define min(A, B)
- #define meminfo() memdump(__FILE__, __LINE__)

Functions

- char * strclone (const char *str)
- void memdump (char *filename, int line)

4.7.1 Macro Definition Documentation

4.7.1.1 max

```
#define max(
               A,
               B )
```

Value:

```
    ({ \
    typeof(A) _tempx = (A); \
    typeof(B) _tempy = (B); \
    _tempx > _tempy ? _tempx : _tempy; \
})
```

Macro that compares two values and returns the greater one.

Parameters

A	The first value.
B	The second value.

Returns

The value that is greater than the other.

Definition at line 11 of file snippets.h.

4.7.1.2 meminfo

```
#define meminfo( ) memdump(__FILE__, __LINE__)
```

Calls memdump() with the current file's name and the line number where the macro was used.

Definition at line 45 of file snippets.h.

4.7.1.3 min

```
#define min(
               A,
               B )
```

Value:

```
    ({ \
    typeof(A) _tempx = (A); \
    typeof(B) _tempy = (B); \
    _tempx < _tempy ? _tempx : _tempy; \
})
```

Macro that compares two values and returns the smaller one.

Parameters

A	The first value.
B	The second value.

Returns

The value that is lesser than the other.

Definition at line 25 of file snippets.h.

4.7.2 Function Documentation

4.7.2.1 memdump()

```
void memdump (
             char * filename,
             int line )
```

Prints the current amount of memory allocation calls and the sum of the allocated bytes.

Can be given a filename and a line number to indicate where it was called.

Parameters

filename	Path to the file.
line	The line number.

Definition at line 15 of file snippets.c.

4.7.2.2 strclone()

```
char* strclone (
```

```
const char * str )
```

Creates a new copy of a given string. The new string should be freed by the caller as it is dynamically allocated.

Parameters

str	The string to make a copy of.

Returns

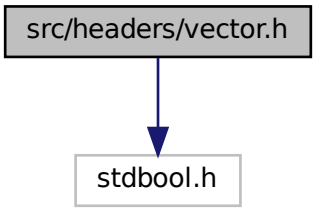
The new string with the same contents as the input.

Definition at line 6 of file snippets.c.

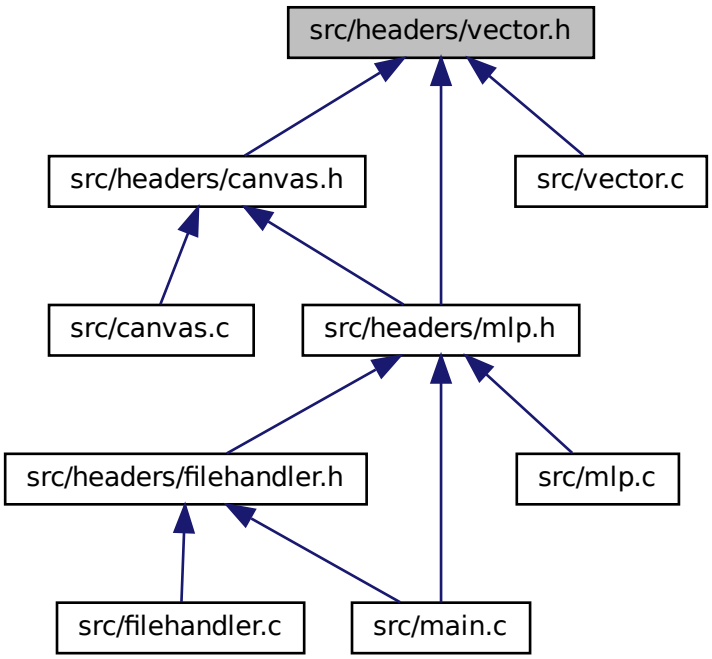
4.8 src/headers/vector.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for vector.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Vector

Macros

- #define SCALING 2
- #define SHRINK 0.2
- #define get_vector_as_type(V, N, T) (∗(T ∗) get_vector_address((V), (N)))

Typedefs

- typedef struct Vector Vector

Functions

- void ∗ get_vector_address (const Vector ∗v, size_t index)
- Vector create_vector (size_t size, size_t elem_size, bool reset)
- void push_vector (Vector ∗v, const void ∗n)
- void pop_vector (Vector ∗v)
- void insert_vector (Vector ∗v, const void ∗n, size_t index)
- void erase_vector (Vector ∗v, size_t index)
- void free_vector (Vector ∗v)

4.8.1 Macro Definition Documentation

4.8.1.1 get_vector_as_type

```
#define get_vector_as_type(
              V,
              N,
              T ) (*(T *) get_vector_address((V), (N)))
```

Queries a Vector at a given index and returns the stored value.

Parameters

V	Pointer to the target Vector.
N	The index.
T	The type of the value to be returned from the Vector.

Returns

The stored value.

Definition at line 27 of file vector.h.

4.8.1.2 SCALING

```
#define SCALING 2
```

Definition at line 5 of file vector.h.

4.8.1.3 SHRINK

```
#define SHRINK 0.2
```

Definition at line 6 of file vector.h.

4.8.2 Typedef Documentation

4.8.2.1 Vector

```
typedef struct Vector Vector
```

Valami leírás

4.8.3 Function Documentation

4.8.3.1 create_vector()

```
Vector create_vector (
            size_t size,
            size_t elem_size,
            bool reset )
```

Creates a new Vector with a given capacity. The returned Vector should be freed by the caller, as it contains dynamically allocated memory.

Parameters

size	Starting capacity of the Vector.
elem_size	The number of bytes to allocate for a single element.
reset	Should the allocation reset all memory-garbage to 0?

Returns

The new Vector struct.

Definition at line 74 of file vector.c.

4.8.3.2 erase_vector()

```
void erase_vector (
            Vector * v,
            size_t index )
```

Removes an element from a Vector at a given index.

Parameters

v	Pointer to the target Vector.
index	The index of the element to be removed.

Definition at line 121 of file vector.c.

Here is the call graph for this function:

4.8.3.3 free_vector()

```
void free_vector (
            Vector * v )
```

Frees all dynamically allocated memory used by a Vector.

Parameters

v	Pointer to the Vector that should be be freed.

Definition at line 133 of file vector.c.

4.8.3.4 get_vector_address()

```
void* get_vector_address (
            const Vector * v,
            size_t index )
```

Queries a Vector at a given index.

Parameters

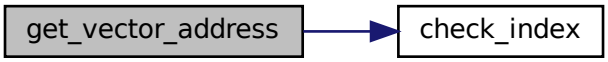
v	Pointer to the target Vector.
index	The index.

Returns

The pointer to where the value's bytes start inside the Vector at the given index.

Definition at line 21 of file vector.c.

Here is the call graph for this function:



4.8.3.5 insert_vector()

```
void insert_vector (
            Vector * v,
            const void * n,
            size_t index )
```

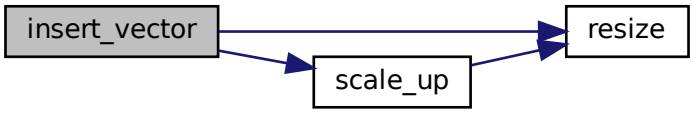
Places a new element into a Vector at a given index.

Parameters

v	Pointer to the target Vector.

Definition at line 103 of file vector.c.

Here is the call graph for this function:



4.8.3.6 pop_vector()

```
void pop_vector (
            Vector * v )
```

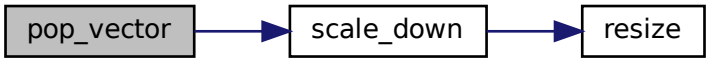
Removes the last element in a Vector.

Parameters

v	Pointer to the target Vector.

Definition at line 93 of file vector.c.

Here is the call graph for this function:



4.8.3.7 push_vector()

```
void push_vector (
            Vector * v,
            const void * n )
```

Places a new element at the end of the Vector.

Parameters

v	Pointer to the target Vector.
n	Pointer to the new element's starting byte.

Definition at line 85 of file vector.c.

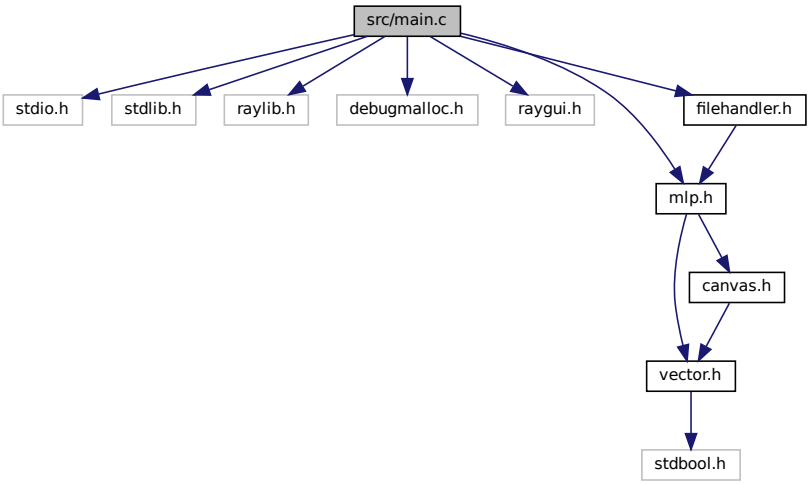
Here is the call graph for this function:

4.9 src/main.c File Reference

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

```
#include "raylib.h"
#include "debugmalloc.h"
#include "raygui.h"
#include "mlp.h"
#include "filehandler.h"
```

Include dependency graph for main.c:



Macros

- #define RAYGUI_IMPLEMENTATION

Functions

- int main ()

4.9.1 Macro Definition Documentation

4.9.1.1 RAYGUI_IMPLEMENTATION

```
#define RAYGUI_IMPLEMENTATION
```

Definition at line 6 of file main.c.

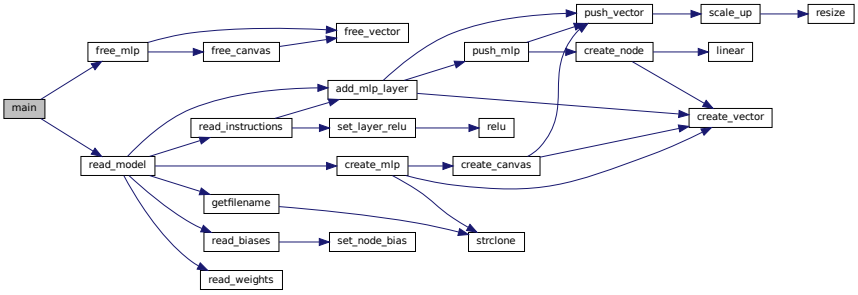
4.9.2 Function Documentation

4.9.2.1 main()

```
int main ( )
```

Definition at line 13 of file main.c.

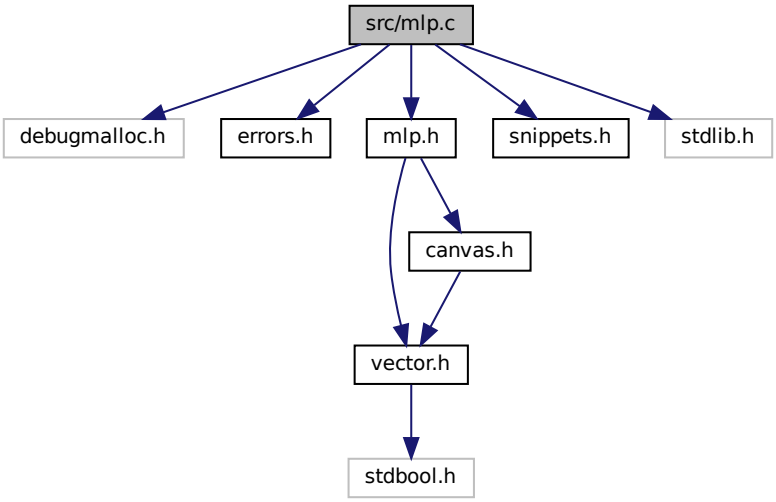
Here is the call graph for this function:



4.10 src/mlp.c File Reference

```
#include "debugmalloc.h"
#include "errors.h"
#include "mlp.h"
#include "snippets.h"
#include <stdlib.h>
```

Include dependency graph for mlp.c:



Functions

- static double relu (double value)
- static double linear (double value)
- static Node create_node (double bias)
- MLP create_mlp (size_t x, size_t y, size_t kx, size_t ky, char *name, size_t layers)
- void add_mlp_layer (MLP *mlp, size_t nodes)
- void set_layer_relu (MLP *mlp, size_t layer)
- void set_layer_linear (MLP *mlp, size_t layer)
- void push_mlp (MLP *mlp, size_t layer, double bias)
- void set_node_bias (MLP *mlp, size_t layer, size_t n, double bias)
- void free_mlp (MLP *mlp)
- static double maxpool2d (MLP *mlp, size_t x, size_t y, size_t width, size_t height)
- void load_mlp_input (MLP *mlp)
- static void act_node (Node *node)
- static void run_node (Vector *layer, Node *node, size_t node_index)
- void run_mlp (MLP *mlp)

4.10.1 Function Documentation

4.10.1.1 act_node()

```
static void act_node (
            Node * node )  [static]
```

Calculates and sets a given Node's output based on its value, bias and activation function.

Parameters

node	Pointer to the target Node.

Definition at line 186 of file mlp.c.

4.10.1.2 add_mlp_layer()

```
void add_mlp_layer (
            MLP * mlp,
            size_t nodes )
```

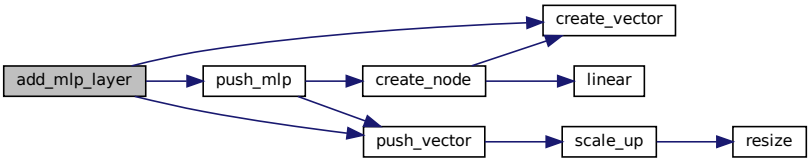
Inserts a new layer at the end of an MLP with a given number of dummy nodes. The new layer will be automatically connected to the layer before it with weights of 1.0 and biases of 0.0.

Parameters

mlp	Pointer to the target MLP.
nodes	Number of dummy nodes.

Definition at line 49 of file mlp.c.

Here is the call graph for this function:



4.10.1.3 create_mlp()

```
MLP create_mlp (
              size_t x,
              size_t y,
              size_t kx,
              size_t ky,
              char * name,
              size_t layers )
```

Creates a new MLP model with a given name and number of layers. The returned MLP should be freed by the caller, as it contains dynamically allocated memory.

Parameters

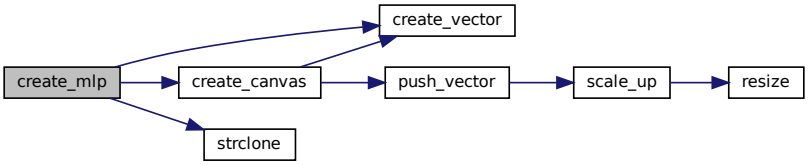
x	Canvas width.
y	Canvas Height.
kx	MaxPool2D kernel width.
ky	MaxPool2D kernel height.
name	String with the name to copy.
layers	Number of layers to start with.

Returns

The newly created MLP struct.

Definition at line 35 of file mlp.c.

Here is the call graph for this function:



4.10.1.4 create_node()

```
static Node create_node (
            double bias )  [static]
```

Creates a new Node with a given bias. The new Node should be freed by the caller as it contains dynamically allocated memory.

Parameters

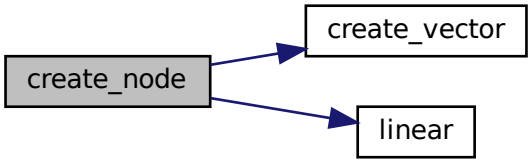
bias	The Node's bias.

Returns

The new Node.

Definition at line 27 of file mlp.c.

Here is the call graph for this function:



4.10.1.5 free_mlp()

```
void free_mlp (
            MLP * mlp )
```

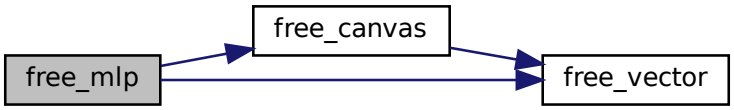
Frees all the dynamically allocated memory used by the model.

Parameters

mlp	Pointer to the MLP.

Definition at line 116 of file mlp.c.

Here is the call graph for this function:



4.10.1.6 linear()

```
static double linear (
            double value )  [static]
```

Definition at line 13 of file mlp.c.

4.10.1.7 load_mlp_input()

```
void load_mlp_input (
            MLP * mlp )
```

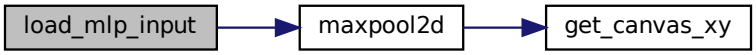
Loads the contents of an MLP's Canvas to the input layer. The MaxPooling is also done by this step.

Parameters

mlp	Pointer to the target MLP.

Definition at line 163 of file mlp.c.

Here is the call graph for this function:



4.10.1.8 maxpool2d()

```
static double maxpool2d (
            MLP * mlp,
            size_t x,
            size_t y,
            size_t width,
            size_t height )  [static]
```

Finds the maximum value inside a given 2D area of an MLP's Canvas.

Parameters

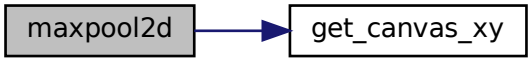
mlp	Pointer to the target MLP.
x	X coordinate of the area's top left corner.
y	Y coordinate of the area's top left corner.
width	The area's width.
height	The area's height.

Returns

The maximum value inside the given area.

Definition at line 149 of file mlp.c.

Here is the call graph for this function:



4.10.1.9 push_mlp()

```
void push_mlp (
            MLP * mlp,
            size_t layer,
            double bias )
```

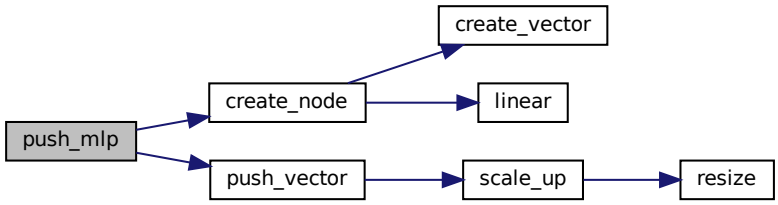
Inserts a new Node at the end of a layer in an MLP. The new Node will have default connections with 1.0 as weight.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.
bias	The new Node's bias.

Definition at line 80 of file mlp.c.

Here is the call graph for this function:



4.10.1.10 relu()

```
static double relu (
            double value )  [static]
```

Definition at line 8 of file mlp.c.

4.10.1.11 run_mlp()

```
void run_mlp (
            MLP * mlp )
```

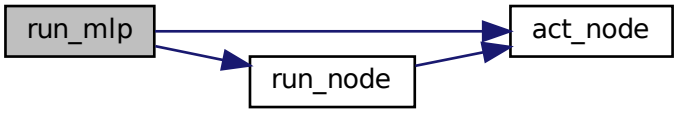
Runs the MLP model in a simple feed-forward manner, calculating the output for the current input layer.

Parameters

mlp	Pointer to the target MLP.

Definition at line 214 of file mlp.c.

Here is the call graph for this function:



4.10.1.12 run_node()

```
static void run_node (
            Vector * layer,
            Node * node,
            size_t node_index )  [static]
```

Calculates and sets a given Node's value and output. The output is based on the Node's value, bias and activation function.

The target Node's value will be the sum of each previous Node's output multiplied by the weight of its connection to the target Node.

Parameters

layer	Pointer to the previous layer inside an MLP.
node	Pointer to the target Node.

Definition at line 203 of file mlp.c.

Here is the call graph for this function:

4.10.1.13 set_layer_linear()

```
void set_layer_linear (
            MLP * mlp,
            size_t layer )
```

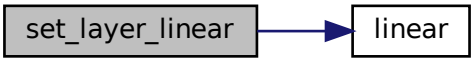
Sets the activation function of all nodes in a layer to linear. The linear function will keep each value as is.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.

Definition at line 69 of file mlp.c.

Here is the call graph for this function:



4.10.1.14 set_layer_relu()

```
void set_layer_relu (
            MLP * mlp,
            size_t layer )
```

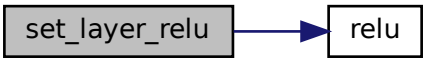
Sets the activation function of all nodes in a layer to ReLU. ReLU will set each negative value to zero and keeps the others.

Parameters

mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.

Definition at line 58 of file mlp.c.

Here is the call graph for this function:



4.10.1.15 set_node_bias()

```
void set_node_bias (
            MLP * mlp,
            size_t layer,
            size_t n,
            double bias )
```

Sets a Node's bias in an MLP.

Parameters

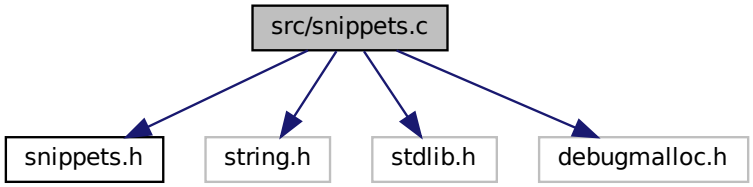
mlp	Pointer to the target MLP.
layer	The layer's index inside the MLP.
n	The Node's index insite the layer.
bias	The Node's new bias.

Definition at line 109 of file mlp.c.

4.11 src/snippets.c File Reference

```
#include "snippets.h"
#include <string.h>
#include <stdlib.h>
#include "debugmalloc.h"
```

Include dependency graph for snippets.c:



Functions

- char ∗ strclone (const char ∗str)
- void memdump (char ∗filename, int line)

4.11.1 Function Documentation

4.11.1.1 memdump()

```
void memdump (
            char * filename,
            int line )
```

Prints the current amount of memory allocation calls and the sum of the allocated bytes.

Can be given a filename and a line number to indicate where it was called.

Parameters

filename	Path to the file.
line	The line number.

Definition at line 15 of file snippets.c.

4.11.1.2 strclone()

```
char* strclone (
            const char * str )
```

Creates a new copy of a given string. The new string should be freed by the caller as it is dynamically allocated.

Parameters

str	The string to make a copy of.

Returns

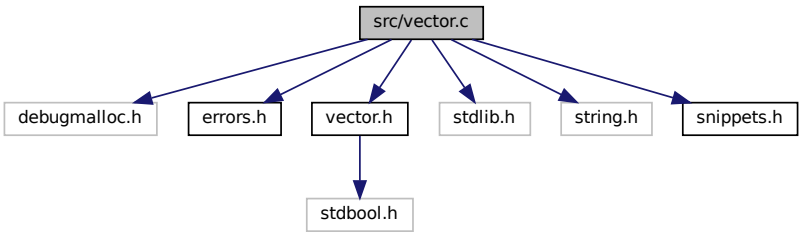
The new string with the same contents as the input.

Definition at line 6 of file snippets.c.

4.12 src/vector.c File Reference

```
#include "debugmalloc.h"
#include "errors.h"
#include "vector.h"
#include <stdlib.h>
#include <string.h>
#include "snippets.h"
```

Include dependency graph for vector.c:



Functions

- static void check_index (const Vector *v, size_t index)
- void * get_vector_address (const Vector *v, size_t index)
- void resize (Vector *v, size_t new_size)
- bool scale_up (Vector *v)
- bool scale_down (Vector *v)
- Vector create_vector (size_t size, size_t elem_size, bool reset)
- void push_vector (Vector *v, const void *n)
- void pop_vector (Vector *v)
- void insert_vector (Vector *v, const void *n, size_t index)
- void erase_vector (Vector *v, size_t index)
- void free_vector (Vector *v)

4.12.1 Function Documentation

4.12.1.1 check_index()

```
static void check_index (
            const Vector * v,
            size_t index )  [static]
```

Terminates the program if an index in a Vector is out of bounds.

Parameters

v	Pointer to the target Vector.
index	The element's index.

Definition at line 15 of file vector.c.

4.12.1.2 create_vector()

```
Vector create_vector (
            size_t size,
            size_t elem_size,
            bool reset )
```

Creates a new Vector with a given capacity. The returned Vector should be freed by the caller, as it contains dynamically allocated memory.

Parameters

size	Starting capacity of the Vector.
elem_size	The number of bytes to allocate for a single element.
reset	Should the allocation reset all memory-garbage to 0?

Returns

The new Vector struct.

Definition at line 74 of file vector.c.

4.12.1.3 erase_vector()

```
void erase_vector (
            Vector * v,
            size_t index )
```

Removes an element from a Vector at a given index.

Parameters

v	Pointer to the target Vector.
index	The index of the element to be removed.

Definition at line 121 of file vector.c.

Here is the call graph for this function:

4.12.1.4 free_vector()

```
void free_vector (
            Vector * v )
```

Frees all dynamically allocated memory used by a Vector.

Parameters

v	Pointer to the Vector that should be be freed.

Definition at line 133 of file vector.c.

4.12.1.5 get_vector_address()

```
void* get_vector_address (
            const Vector * v,
            size_t index )
```

Queries a Vector at a given index.

Parameters

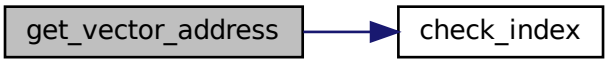
v	Pointer to the target Vector.
index	The index.

Returns

The pointer to where the value's bytes start inside the Vector at the given index.

Definition at line 21 of file vector.c.

Here is the call graph for this function:



4.12.1.6 insert_vector()

```
void insert_vector (
            Vector * v,
            const void * n,
            size_t index )
```

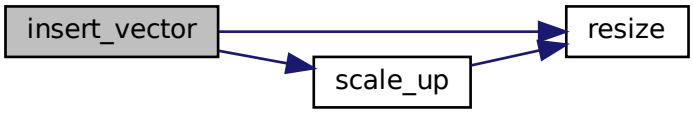
Places a new element into a Vector at a given index.

Parameters

v	Pointer to the target Vector.

Definition at line 103 of file vector.c.

Here is the call graph for this function:



4.12.1.7 pop_vector()

```
void pop_vector (
            Vector * v )
```

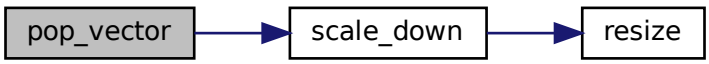
Removes the last element in a Vector.

Parameters

v	Pointer to the target Vector.

Definition at line 93 of file vector.c.

Here is the call graph for this function:



4.12.1.8 push_vector()

```
void push_vector (
            Vector * v,
            const void * n )
```

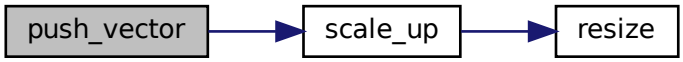
Places a new element at the end of the Vector.

Parameters

v	Pointer to the target Vector.
n	Pointer to the new element's starting byte.

Definition at line 85 of file vector.c.

Here is the call graph for this function:



4.12.1.9 resize()

```
void resize (
             Vector * v,
             size_t new_size )
```

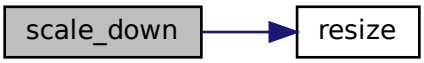
Definition at line 28 of file vector.c.

4.12.1.10 scale_down()

```
bool scale_down (
             Vector * v )
```

Definition at line 55 of file vector.c.

Here is the call graph for this function:



4.12.1.11 scale_up()

```
bool scale_up (
            Vector * v )
```

Definition at line 39 of file vector.c.

Here is the call graph for this function:

Index