

Rajzfelismerő

Generated by Doxygen 1.9.1

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 Canvas Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 m	5
3.2 MLP Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Field Documentation	6
3.2.2.1 canvas	6
3.2.2.2 kx	6
3.2.2.3 ky	6
3.2.2.4 layers	7
3.2.2.5 name	7
3.2.2.6 x	7
3.2.2.7 y	7
3.3 Node Struct Reference	7
3.3.1 Detailed Description	8
3.3.2 Field Documentation	8
3.3.2.1 act	8
3.3.2.2 bias	8
3.3.2.3 con	8
3.3.2.4 output	8
3.3.2.5 value	9
3.4 ReadResult Struct Reference	9
3.4.1 Detailed Description	9
3.4.2 Field Documentation	9
3.4.2.1 model	9
3.4.2.2 status	9
3.5 Vector Struct Reference	10
3.5.1 Detailed Description	10
3.5.2 Field Documentation	10
3.5.2.1 arr	10
3.5.2.2 cap	10
3.5.2.3 elem_size	10
3.5.2.4 size	10
4 File Documentation	11

4.1 src/canvas.c File Reference	11
4.1.1 Function Documentation	12
4.1.1.1 create_canvas()	12
4.1.1.2 free_canvas()	12
4.1.1.3 get_canvas_xy()	13
4.2 src/filehandler.c File Reference	13
4.2.1 Macro Definition Documentation	14
4.2.1.1 pass [1/2]	14
4.2.1.2 pass [2/2]	15
4.2.2 Function Documentation	15
4.2.2.1 getfilename()	15
4.2.2.2 read_biases()	15
4.2.2.3 read_instructions()	16
4.2.2.4 read_model()	17
4.2.2.5 read_weights()	18
4.3 src/headers/canvas.h File Reference	18
4.3.1 Typedef Documentation	19
4.3.1.1 Canvas	19
4.3.2 Function Documentation	20
4.3.2.1 create_canvas()	20
4.3.2.2 free_canvas()	20
4.3.2.3 get_canvas_xy()	21
4.4 src/headers/errors.h File Reference	22
4.4.1 Typedef Documentation	22
4.4.1.1 ERROR	22
4.4.2 Enumeration Type Documentation	22
4.4.2.1 ERROR	22
4.5 src/headers/filehandler.h File Reference	23
4.5.1 Typedef Documentation	24
4.5.1.1 ReadResult	24
4.5.1.2 RSTATUS	25
4.5.2 Enumeration Type Documentation	25
4.5.2.1 RSTATUS	25
4.5.3 Function Documentation	25
4.5.3.1 read_model()	25
4.6 src/headers/mlp.h File Reference	26
4.6.1 Typedef Documentation	27
4.6.1.1 MLP	27
4.6.1.2 Node	28
4.6.2 Function Documentation	28
4.6.2.1 add_mlp_layer()	28
4.6.2.2 create_mlp()	28

4.6.2.3 free_mlp()	29
4.6.2.4 load_mlp_input()	30
4.6.2.5 push_mlp()	30
4.6.2.6 run_mlp()	31
4.6.2.7 set_layer_linear()	32
4.6.2.8 set_layer_relu()	32
4.6.2.9 set_node_bias()	33
4.7 src/headers/snippets.h File Reference	33
4.7.1 Macro Definition Documentation	34
4.7.1.1 max	34
4.7.1.2 meminfo	34
4.7.1.3 min	35
4.7.2 Function Documentation	35
4.7.2.1 memdump()	35
4.7.2.2 strclone()	35
4.8 src/headers/vector.h File Reference	36
4.8.1 Macro Definition Documentation	38
4.8.1.1 get_vector_as_type	38
4.8.1.2 SCALING	38
4.8.1.3 SHRINK	38
4.8.2 Typedef Documentation	38
4.8.2.1 Vector	39
4.8.3 Function Documentation	39
4.8.3.1 create_vector()	39
4.8.3.2 erase_vector()	39
4.8.3.3 free_vector()	40
4.8.3.4 get_vector_address()	40
4.8.3.5 insert_vector()	41
4.8.3.6 pop_vector()	41
4.8.3.7 push_vector()	42
4.9 src/main.c File Reference	42
4.9.1 Macro Definition Documentation	43
4.9.1.1 RAYGUI_IMPLEMENTATION	43
4.9.2 Function Documentation	43
4.9.2.1 main()	44
4.10 src/mlp.c File Reference	44
4.10.1 Function Documentation	45
4.10.1.1 act_node()	45
4.10.1.2 add_mlp_layer()	45
4.10.1.3 create_mlp()	46
4.10.1.4 create_node()	47
4.10.1.5 free_mlp()	48

4.10.1.6 linear()	48
4.10.1.7 load_mlp_input()	48
4.10.1.8 maxpool2d()	49
4.10.1.9 push_mlp()	50
4.10.1.10 relu()	50
4.10.1.11 run_mlp()	50
4.10.1.12 run_node()	51
4.10.1.13 set_layer_linear()	52
4.10.1.14 set_layer_relu()	52
4.10.1.15 set_node_bias()	53
4.11 src/snippets.c File Reference	53
4.11.1 Function Documentation	54
4.11.1.1 memdump()	54
4.11.1.2 strclone()	54
4.12 src/vector.c File Reference	55
4.12.1 Function Documentation	55
4.12.1.1 check_index()	56
4.12.1.2 create_vector()	56
4.12.1.3 erase_vector()	56
4.12.1.4 free_vector()	57
4.12.1.5 get_vector_address()	57
4.12.1.6 insert_vector()	58
4.12.1.7 pop_vector()	59
4.12.1.8 push_vector()	59
4.12.1.9 resize()	60
4.12.1.10 scale_down()	60
4.12.1.11 scale_up()	61
Index	63

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Canvas	5
MLP	6
Node	7
ReadResult	9
Vector	10

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/ canvas.c	11
src/ filehandler.c	13
src/ main.c	42
src/ mlp.c	44
src/ snippets.c	53
src/ vector.c	55
src/headers/ canvas.h	18
src/headers/ errors.h	22
src/headers/ filehandler.h	23
src/headers/ mlp.h	26
src/headers/ snippets.h	33
src/headers/ vector.h	36

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 successfully. 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:

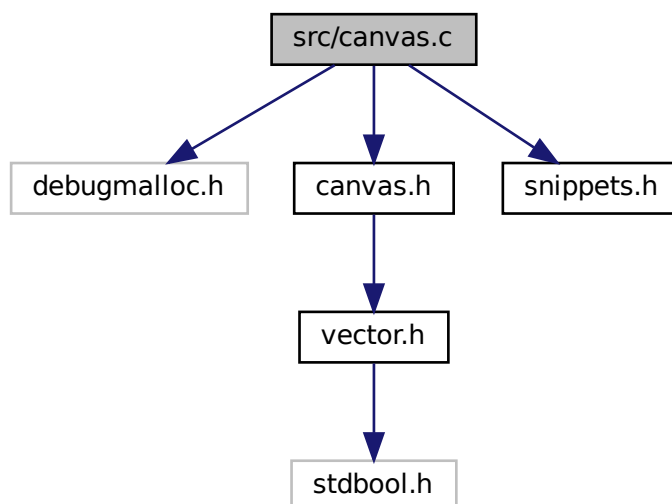
- src/headers/[vector.h](#)

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

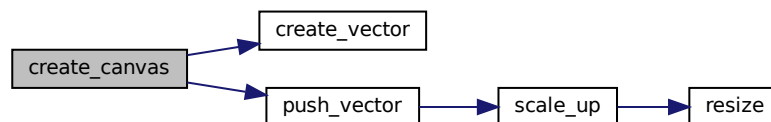
<i>width</i>	The width of the canvas.
<i>height</i>	The height of the canvas.

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

<i>canvas</i>	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

<i>canvas</i>	Pointer to the target canvas.
<i>x</i>	The X coordinate on the canvas.
<i>y</i>	The Y coordinate on the canvas.

Returns

The greyscale value on the canvas at the given coordinates.

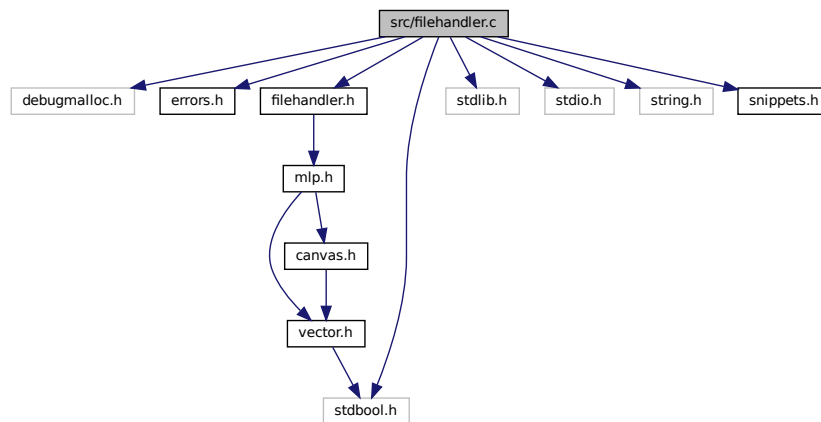
Definition at line 28 of file canvas.c.

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 caller as it's dynamically allocated.

Parameters

<i>path</i>	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

<i>f</i>	The file to read from.
<i>mlp</i>	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

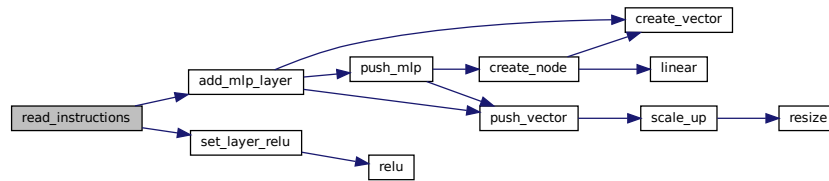
<i>f</i>	The file to read from.
<i>mlp</i>	Pointer to the target MLP .
<i>n</i>	Number of instructions.

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

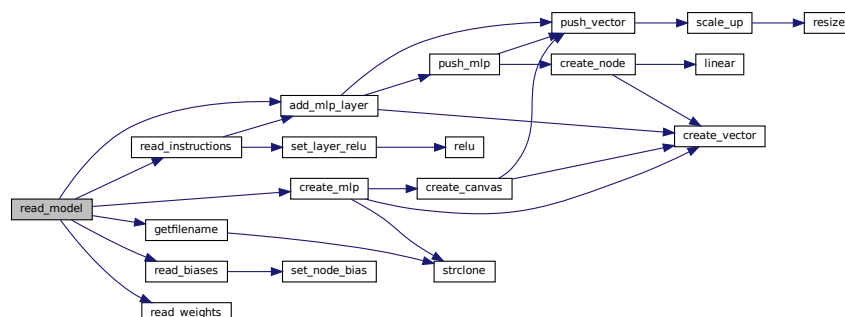
<i>path</i>	The path to the file.
-------------	-----------------------

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

<i>f</i>	The file to read from.
<i>mlp</i>	Pointer to the target MLP .

Returns

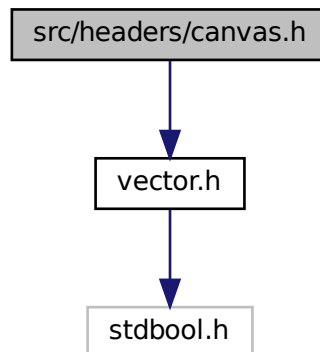
An RSTATUS with the possible status codes.

Definition at line 113 of file filehandler.c.

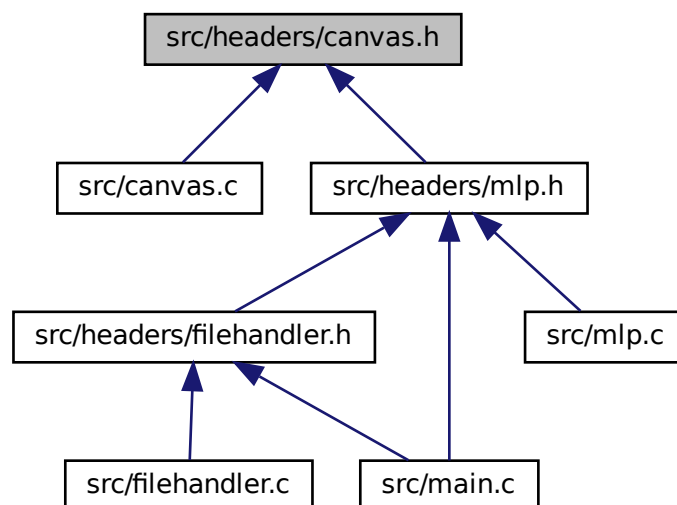
4.3 src/headers/canvas.h File Reference

```
#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

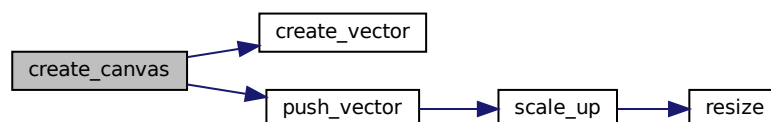
<i>width</i>	The width of the canvas.
<i>height</i>	The height of the canvas.

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

<i>canvas</i>	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

<i>canvas</i>	Pointer to the target canvas.
<i>x</i>	The X coordinate on the canvas.
<i>y</i>	The Y coordinate on the canvas.

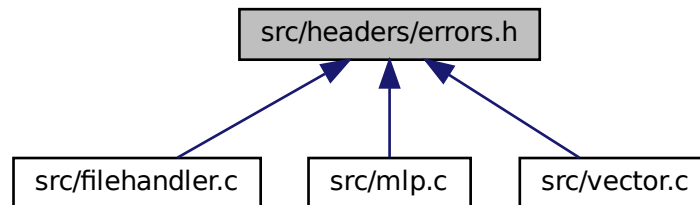
Returns

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

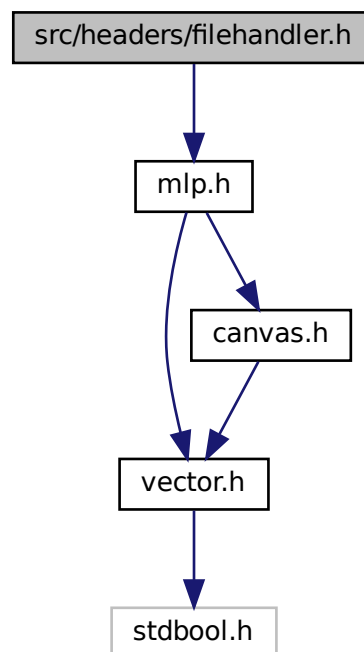
ERR_NULLPOINTER	
ERR_INDEXOUTOFBOUNDS	

Definition at line 6 of file errors.h.

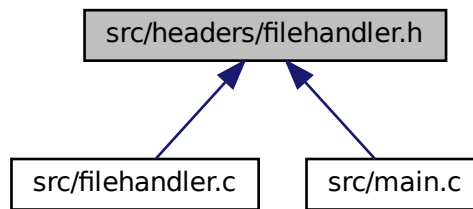
4.5 src/headers/filehandler.h File Reference

```
#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

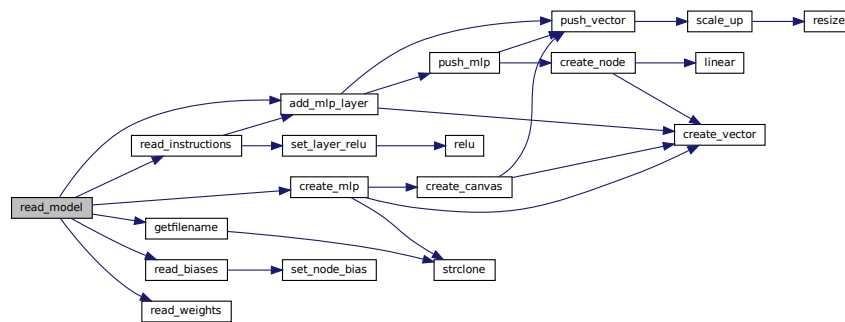
<i>path</i>	The path to the file.
-------------	-----------------------

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.

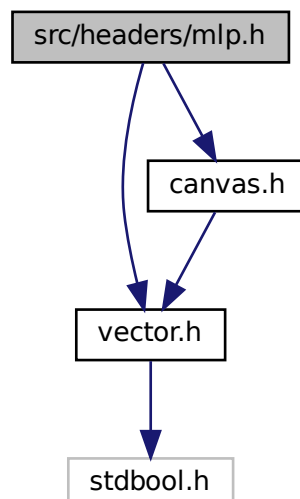
Definition at line 136 of file filehandler.c.

Here is the call graph for this function:

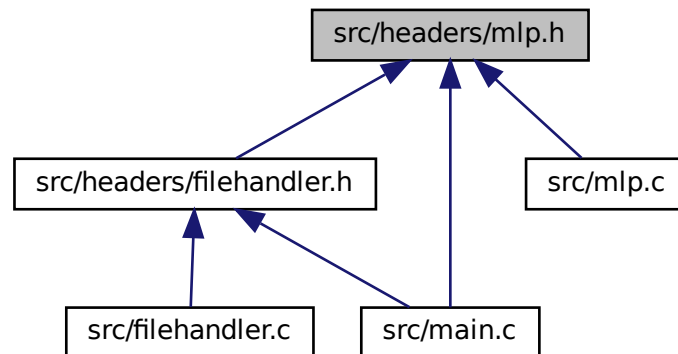


4.6 src/headers/mlp.h File Reference

```
#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 )
```

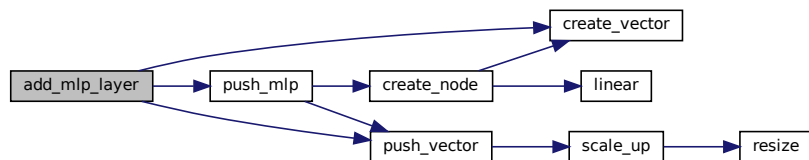
Inserts a new layer at the end of an [MLP](#) with a given number of dummy nodes. The new layer will be automatically connected to the layer before it with weights of 1.0 and biases of 0.0.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>nodes</i>	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

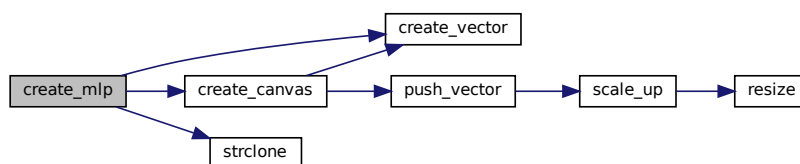
<i>x</i>	Canvas width.
<i>y</i>	Canvas Height.
<i>kx</i>	MaxPool2D kernel width.
<i>ky</i>	MaxPool2D kernel height.
<i>name</i>	String with the name to copy.
<i>layers</i>	Number of layers to start with.

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 )
```

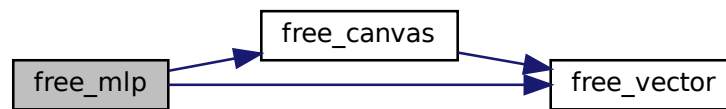
Frees all the dynamically allocated memory used by the model.

Parameters

<i>mlp</i>	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 )
```

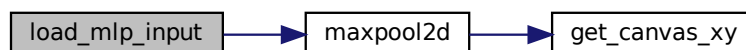
Loads the contents of an [MLP's Canvas](#) to the input layer. The MaxPooling is also done by this step.

Parameters

<i>mlp</i>	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 )
```

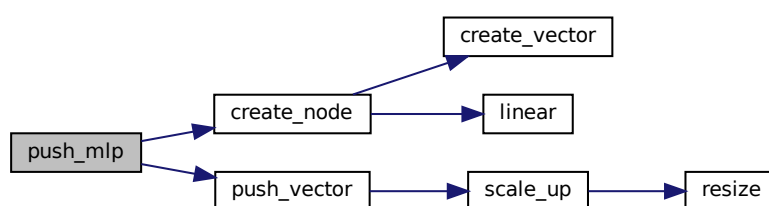
Inserts a new [Node](#) at the end of a layer in an [MLP](#). The new [Node](#) will have default connections with 1.0 as weight.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	The layer's index inside the MLP .
<i>bias</i>	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 )
```

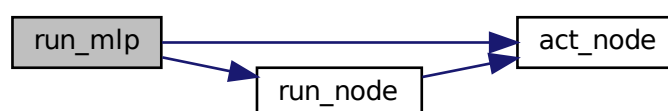
Runs the [MLP](#) model in a simple feed-forward manner, calculating the output for the current input layer.

Parameters

<i>mlp</i>	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 )
```

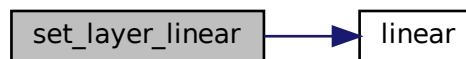
Sets the activation function of all nodes in a layer to linear. The linear function will keep each value as is.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	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 )
```

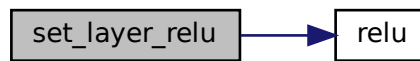
Sets the activation function of all nodes in a layer to ReLU. ReLU will set each negative value to zero and keeps the others.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	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](#).

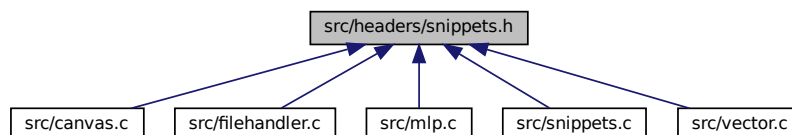
Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	The layer's index inside the MLP .
<i>n</i>	The Node 's index insite the layer.
<i>bias</i>	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

<i>A</i>	The first value.
<i>B</i>	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

<i>A</i>	The first value.
<i>B</i>	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

<i>filename</i>	Path to the file.
<i>line</i>	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

<i>str</i>	The string to make a copy of.
------------	-------------------------------

Returns

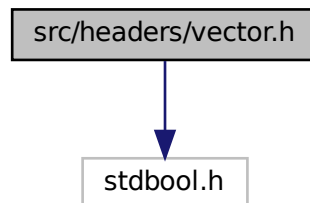
The new string with the same contents as the input.

Definition at line 6 of file snippets.c.

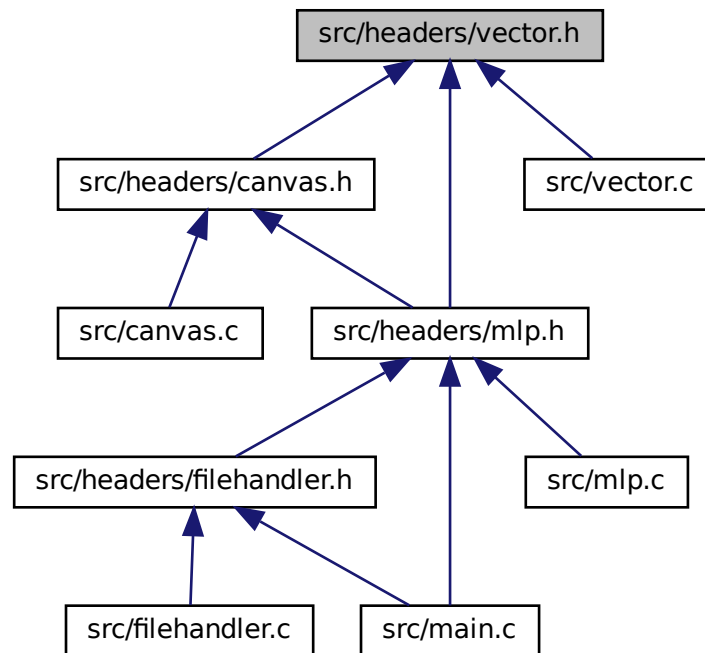
4.8 src/headers/vector.h File Reference

```
#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

<i>V</i>	Pointer to the target Vector .
<i>N</i>	The index.
<i>T</i>	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

<i>size</i>	Starting capacity of the Vector .
<i>elem_size</i>	The number of bytes to allocate for a single element.
<i>reset</i>	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

<i>v</i>	Pointer to the target Vector .
<i>index</i>	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

<i>v</i>	Pointer to the Vector that should 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

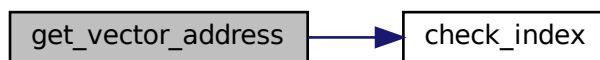
<i>v</i>	Pointer to the target Vector .
<i>index</i>	The index.

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 )
```

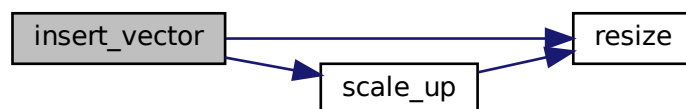
Places a new element into a [Vector](#) at a given index.

Parameters

<i>v</i>	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 )
```

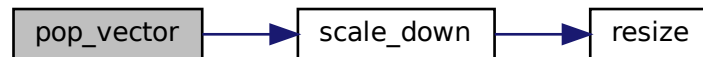
Removes the last element in a [Vector](#).

Parameters

<i>v</i>	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 )
```

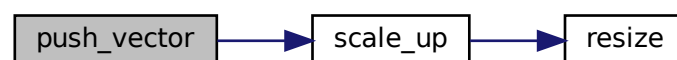
Places a new element at the end of the [Vector](#).

Parameters

<i>v</i>	Pointer to the target Vector .
<i>n</i>	Pointer to the new element's starting byte.

Definition at line 85 of file vector.c.

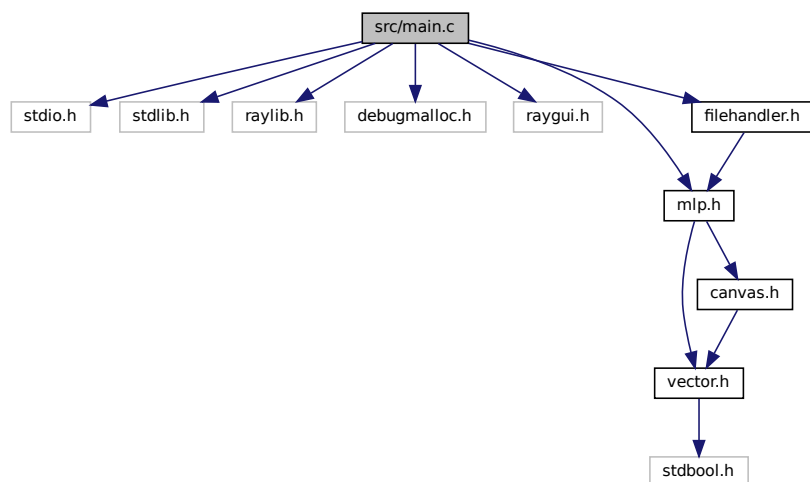
Here is the call graph for this function:

**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`.

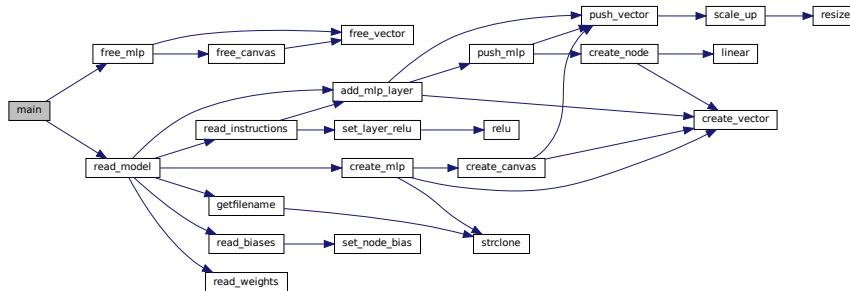
4.9.2 Function Documentation

4.9.2.1 main()

```
int main ( )
```

Definition at line 13 of file main.c.

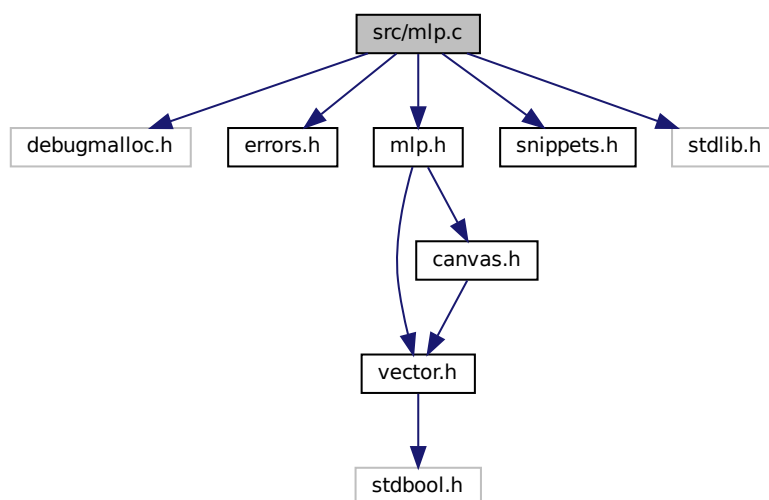
Here is the call graph for this function:



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

<i>node</i>	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 )
```

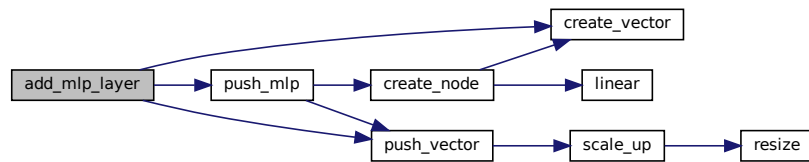
Inserts a new layer at the end of an [MLP](#) with a given number of dummy nodes. The new layer will be automatically connected to the layer before it with weights of 1.0 and biases of 0.0.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>nodes</i>	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

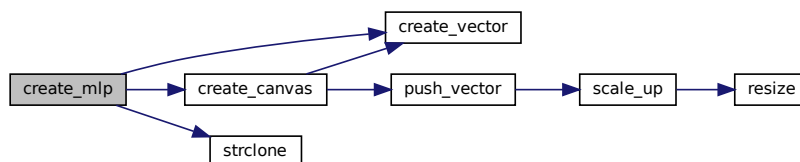
<i>x</i>	Canvas width.
<i>y</i>	Canvas Height.
<i>kx</i>	MaxPool2D kernel width.
<i>ky</i>	MaxPool2D kernel height.
<i>name</i>	String with the name to copy.
<i>layers</i>	Number of layers to start with.

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

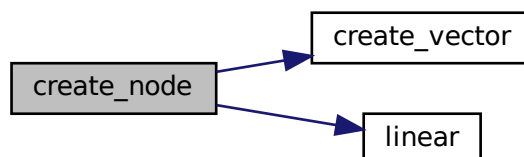
<i>bias</i>	The Node 's bias.
-------------	-----------------------------------

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 )
```

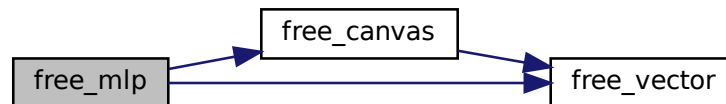
Frees all the dynamically allocated memory used by the model.

Parameters

<i>mlp</i>	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 )
```

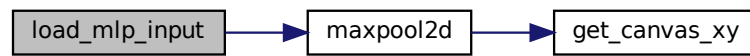
Loads the contents of an MLP's Canvas to the input layer. The MaxPooling is also done by this step.

Parameters

<i>mlp</i>	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

<i>mlp</i>	Pointer to the target MLP .
<i>x</i>	X coordinate of the area's top left corner.
<i>y</i>	Y coordinate of the area's top left corner.
<i>width</i>	The area's width.
<i>height</i>	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 )
```

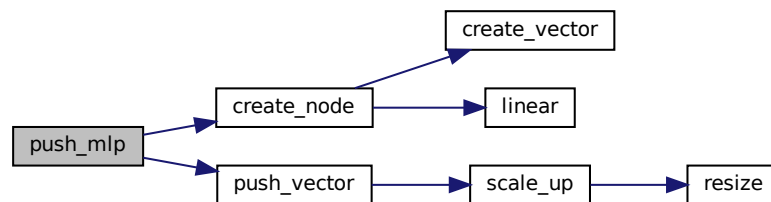
Inserts a new [Node](#) at the end of a layer in an [MLP](#). The new [Node](#) will have default connections with 1.0 as weight.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	The layer's index inside the MLP .
<i>bias</i>	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 )
```

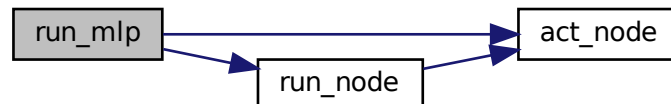
Runs the [MLP](#) model in a simple feed-forward manner, calculating the output for the current input layer.

Parameters

<i>mlp</i>	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

<i>layer</i>	Pointer to the previous layer inside an MLP .
<i>node</i>	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 )
```

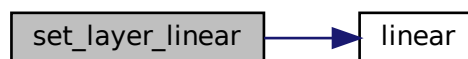
Sets the activation function of all nodes in a layer to linear. The linear function will keep each value as is.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	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 )
```

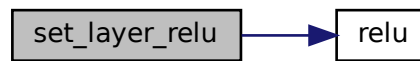
Sets the activation function of all nodes in a layer to ReLU. ReLU will set each negative value to zero and keeps the others.

Parameters

<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	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

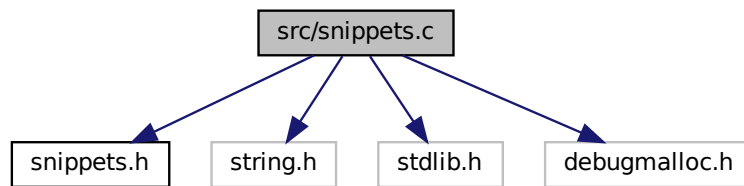
<i>mlp</i>	Pointer to the target MLP .
<i>layer</i>	The layer's index inside the MLP .
<i>n</i>	The Node 's index insite the layer.
<i>bias</i>	The Node 's new bias.

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

<i>filename</i>	Path to the file.
<i>line</i>	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

<i>str</i>	The string to make a copy of.
------------	-------------------------------

Returns

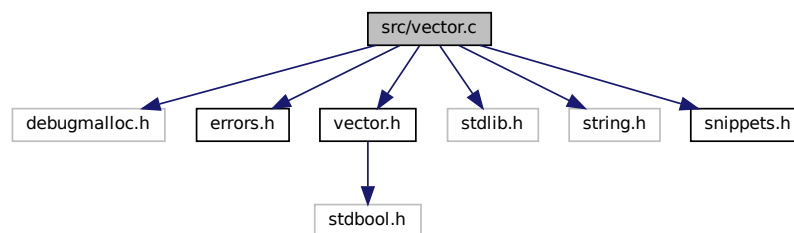
The new string with the same contents as the input.

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

<i>v</i>	Pointer to the target Vector .
<i>index</i>	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

<i>size</i>	Starting capacity of the Vector .
<i>elem_size</i>	The number of bytes to allocate for a single element.
<i>reset</i>	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

<i>v</i>	Pointer to the target Vector .
<i>index</i>	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

<i>v</i>	Pointer to the Vector that should 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

<i>v</i>	Pointer to the target Vector .
<i>index</i>	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 )
```

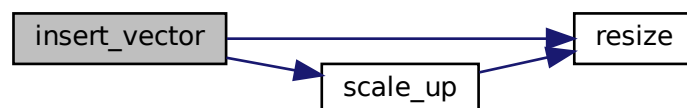
Places a new element into a [Vector](#) at a given index.

Parameters

<i>v</i>	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 )
```

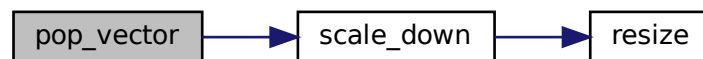
Removes the last element in a [Vector](#).

Parameters

<i>v</i>	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

<i>v</i>	Pointer to the target Vector .
<i>n</i>	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

- act
 - Node, [8](#)
- act_node
 - mlp.c, [45](#)
- add_mlp_layer
 - mlp.c, [45](#)
 - mlp.h, [28](#)
- arr
 - Vector, [10](#)
- bias
 - Node, [8](#)
- Canvas, [5](#)
 - canvas.h, [19](#)
 - m, [5](#)
- canvas
 - MLP, [6](#)
- canvas.c
 - create_canvas, [12](#)
 - free_canvas, [12](#)
 - get_canvas_xy, [13](#)
- canvas.h
 - Canvas, [19](#)
 - create_canvas, [20](#)
 - free_canvas, [20](#)
 - get_canvas_xy, [21](#)
- cap
 - Vector, [10](#)
- check_index
 - vector.c, [55](#)
- con
 - Node, [8](#)
- create_canvas
 - canvas.c, [12](#)
 - canvas.h, [20](#)
- create_mlp
 - mlp.c, [46](#)
 - mlp.h, [28](#)
- create_node
 - mlp.c, [47](#)
- create_vector
 - vector.c, [56](#)
 - vector.h, [39](#)
- elem_size
 - Vector, [10](#)
- erase_vector
 - vector.c, [56](#)
 - vector.h, [39](#)
- ERR_INDEXOUTOFBOUNDS
 - errors.h, [23](#)
- ERR_NULLPOINTER
 - errors.h, [23](#)
- ERROR
 - errors.h, [22](#)
- errors.h
 - ERR_INDEXOUTOFBOUNDS, [23](#)
 - ERR_NULLPOINTER, [23](#)
 - ERROR, [22](#)
- filehandler.c
 - getfilename, [15](#)
 - pass, [14](#)
 - read_biases, [15](#)
 - read_instructions, [16](#)
 - read_model, [17](#)
 - read_weights, [17](#)
- filehandler.h
 - KERNELSIZE, [25](#)
 - NODATA, [25](#)
 - NOFILE, [25](#)
 - NOLAYER, [25](#)
 - read_model, [25](#)
 - ReadResult, [24](#)
 - RSTATUS, [24](#), [25](#)
 - SUCCESS, [25](#)
 - WRONGINSTRUCTION, [25](#)
- free_canvas
 - canvas.c, [12](#)
 - canvas.h, [20](#)
- free_mlp
 - mlp.c, [47](#)
 - mlp.h, [29](#)
- free_vector
 - vector.c, [57](#)
 - vector.h, [40](#)
- get_canvas_xy
 - canvas.c, [13](#)
 - canvas.h, [21](#)
- get_vector_address
 - vector.c, [57](#)
 - vector.h, [40](#)
- get_vector_as_type
 - vector.h, [38](#)
- getfilename
 - filehandler.c, [15](#)
- insert_vector

- vector.c, 58
- vector.h, 41
- KERNELSIZE
 - filehandler.h, 25
- kx
 - MLP, 6
- ky
 - MLP, 6
- layers
 - MLP, 6
- linear
 - mlp.c, 48
- load_mlp_input
 - mlp.c, 48
 - mlp.h, 30
- m
 - Canvas, 5
- main
 - main.c, 43
- main.c
 - main, 43
 - RAYGUI_IMPLEMENTATION, 43
- max
 - snippets.h, 34
- maxpool2d
 - mlp.c, 49
- memdump
 - snippets.c, 54
 - snippets.h, 35
- meminfo
 - snippets.h, 34
- min
 - snippets.h, 34
- MLP, 6
 - canvas, 6
 - kx, 6
 - ky, 6
 - layers, 6
 - mlp.h, 27
 - name, 7
 - x, 7
 - y, 7
- mlp.c
 - act_node, 45
 - add_mlp_layer, 45
 - create_mlp, 46
 - create_node, 47
 - free_mlp, 47
 - linear, 48
 - load_mlp_input, 48
 - maxpool2d, 49
 - push_mlp, 49
 - relu, 50
 - run_mlp, 50
 - run_node, 51
 - set_layer_linear, 52
 - set_layer_relu, 52
 - set_node_bias, 53
- mlp.h
 - add_mlp_layer, 28
 - create_mlp, 28
 - free_mlp, 29
 - load_mlp_input, 30
 - MLP, 27
 - Node, 27
 - push_mlp, 30
 - run_mlp, 31
 - set_layer_linear, 31
 - set_layer_relu, 32
 - set_node_bias, 33
- model
 - ReadResult, 9
- name
 - MLP, 7
- NODATA
 - filehandler.h, 25
- Node, 7
 - act, 8
 - bias, 8
 - con, 8
 - mlp.h, 27
 - output, 8
 - value, 8
- NOFILE
 - filehandler.h, 25
- NOLAYER
 - filehandler.h, 25
- output
 - Node, 8
- pass
 - filehandler.c, 14
- pop_vector
 - vector.c, 58
 - vector.h, 41
- push_mlp
 - mlp.c, 49
 - mlp.h, 30
- push_vector
 - vector.c, 59
 - vector.h, 42
- RAYGUI_IMPLEMENTATION
 - main.c, 43
- read_biases
 - filehandler.c, 15
- read_instructions
 - filehandler.c, 16
- read_model
 - filehandler.c, 17
 - filehandler.h, 25
- read_weights
 - filehandler.c, 17

- ReadResult, 9
 - filehandler.h, 24
 - model, 9
 - status, 9
- relu
 - mlp.c, 50
- resize
 - vector.c, 60
- RSTATUS
 - filehandler.h, 24, 25
- run_mlp
 - mlp.c, 50
 - mlp.h, 31
- run_node
 - mlp.c, 51
- scale_down
 - vector.c, 60
- scale_up
 - vector.c, 60
- SCALING
 - vector.h, 38
- set_layer_linear
 - mlp.c, 52
 - mlp.h, 31
- set_layer_relu
 - mlp.c, 52
 - mlp.h, 32
- set_node_bias
 - mlp.c, 53
 - mlp.h, 33
- SHRINK
 - vector.h, 38
- size
 - Vector, 10
- snippets.c
 - memdump, 54
 - strclone, 54
- snippets.h
 - max, 34
 - memdump, 35
 - meminfo, 34
 - min, 34
 - strclone, 35
- src/canvas.c, 11
- src/filehandler.c, 13
- src/headers/canvas.h, 18
- src/headers/errors.h, 22
- src/headers/filehandler.h, 23
- src/headers/mlp.h, 26
- src/headers/snippets.h, 33
- src/headers/vector.h, 36
- src/main.c, 42
- src/mlp.c, 44
- src/snippets.c, 53
- src/vector.c, 55
- status
 - ReadResult, 9
- strclone
 - snippets.c, 54
 - snippets.h, 35
- SUCCESS
 - filehandler.h, 25
- value
 - Node, 8
- Vector, 10
 - arr, 10
 - cap, 10
 - elem_size, 10
 - size, 10
 - vector.h, 38
- vector.c
 - check_index, 55
 - create_vector, 56
 - erase_vector, 56
 - free_vector, 57
 - get_vector_address, 57
 - insert_vector, 58
 - pop_vector, 58
 - push_vector, 59
 - resize, 60
 - scale_down, 60
 - scale_up, 60
- vector.h
 - create_vector, 39
 - erase_vector, 39
 - free_vector, 40
 - get_vector_address, 40
 - get_vector_as_type, 38
 - insert_vector, 41
 - pop_vector, 41
 - push_vector, 42
 - SCALING, 38
 - SHRINK, 38
 - Vector, 38
- WRONGINSTRUCTION
 - filehandler.h, 25
- x
 - MLP, 7
- y
 - MLP, 7