Generic helper / demonstration library

Generated by Doxygen 1.9.1

Chapter 1

GenericHelpers

This library exists to handle a wide array of things that I've found are fairly common to things I do on a day to day basis. I welcome any contributions to this library - I simply ask that they be written in such a way as to be as generic as possible, so they can be used.

All contributed code *MUST* be documented using DoxyGen typical format as this library is also created to help people learn and understand.

Because of the doxygen configurations building this requires having texlive and pdflatex installed (On Ubuntu texlive-latext-base texlive-latex-extra texlive-latex-recommended and pdflatex packages)

To build it is suggested using the build_with_docs.sh script that will re-generate all the documentation, saving having to run make in docs/latex as described below.

In docs/latex there is a Makefile, and running make in docs/latex after running ninja will generate a refman.pdf pdf reference manual. Note that this requires pdflatex to be installed

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 bgp_ipv4_prefix Struct Reference

This structure contains a prefix (in network byte order) and a CIDR for the prefix.

`#include <decode_helpers.h>`

Data Fields

- __u32 prefix
- __u8 cidr
- __u8 * next

4.1.1 Detailed Description

This structure contains a prefix (in network byte order) and a CIDR for the prefix.

4.1.2 Field Documentation

4.1.2.1 cidr

`__u8 bgp_ipv4_prefix::cidr`

4.1.2.2 next

`__u8* bgp_ipv4_prefix::next`

The cidr of the prefix Pointer to the next entry NLRI entry

4.1.2.3 prefix

```
__u32 bgp_ipv4_prefix::prefix
```

The network byte order prefix read from a BGP NLRI

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

- decode_helpers.h

4.2 data Struct Reference

Data Fields

- __u32 prefix
- __u8 cidr

4.2.1 Field Documentation

4.2.1.1 cidr

```
__u8 data::cidr
```

4.2.1.2 prefix

```
__u32 data::prefix
```

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

- tests/test_linked_list.c

4.3 proto_msg Struct Reference

This structure contains elements used after decoding a protobuf string.

```
#include <decode_helpers.h>
```

Data Fields

- __u64 result
- __u64 msg_code
- __u64 msg_type

4.3.1 Detailed Description

This structure contains elements used after decoding a protobuf string.

4.3.2 Field Documentation

4.3.2.1 msg_code

```
__u64 proto_msg::msg_code
```

The message code derived from the result

4.3.2.2 msg_type

```
__u64 proto_msg::msg_type
```

The message type derived from the result

4.3.2.3 result

```
__u64 proto_msg::result
```

The 64 bit variable integer decode

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

- decode_helpers.h

4.4 route4tree Struct Reference

The route4tree structure is used in the creation of a 32 bit binary tree.

```
#include <linked_list.h>
```

Collaboration diagram for route4tree:

Data Fields

- struct route4tree * set
- struct route4tree * unset
- struct route4tree * parent
- void * data

4.4.1 Detailed Description

The route4tree structure is used in the creation of a 32 bit binary tree.

4.4.2 Field Documentation

4.4.2.1 data

```
void* route4tree::data
```

Void pointer to data to be stored in the tree

4.4.2.2 parent

```
struct route4tree* route4tree::parent
```

Pointer to the parent entry of the tree

4.4.2.3 set

```
struct route4tree* route4tree::set
```

Pointer used for when a binary bit is set

4.4.2.4 unset

```
struct route4tree* route4tree::unset
```

Pointer used for when a binary bit is not set

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

- linked_list.h

Chapter 5

File Documentation

5.1 accelerations.c File Reference

Functions

- int search_array_32 (void *haystack, __u64 offset, size_t struct_size, int array_size, __u32 needle)

5.1.1 Function Documentation

5.1.1.1 search_array_32()

```
int search_array_32 (
            void * haystack,
            __u64 offset,
            size_t struct_size,
            int array_size,
            __u32 needle )
```

5.2 accelerations.h File Reference

Functions

- int search_array_32 (void *haystack, __u64 offset, size_t struct_size, size_t array_size, __u32 needle)

  Search an array of structures for a particular 32 bit integer.

5.2.1 Function Documentation

5.2.1.1 search_array_32()

```
int search_array_32 (
            void * haystack,
            __u64 offset,
            size_t struct_size,
            size_t array_size,
            __u32 needle )
```

Search an array of structures for a particular 32 bit integer.

This function takes a pointer to an array of structures and an offset from the base struct to the 32 element in the structure, and then searches the array of structures for the first element of the array where *(struct_ptr+offset) = needle.

NOTE: For this function to work properly the number of array elements has to be divisible by 8

By way of example if we have the following

struct x { __u32 a; __u32 b; __u32 c; } struct x array[24]; If we wanted to search for the first entry where x.b = 10, we would pass the function with the following arguments: array - which would be the pointer to the start of the array, 4 (element b is 4 bytes into the structure), sizeof(struct x) - the size of structure for loop increments, 24 - The number of elements in the array 10 - the needle we are searching for

Parameters

in	*haystack*	- Pointer to the start of the array
in	*offset*	- An offset of the struct to the 32bit element we are searching
in	*struct_size*	- The size of each struct in the struct array
in	*array_size*	- The number of elements in the array
in	*needle*	- The needle we are searching for

Returns

The array offset for the first matching instance or -1 if not found

5.3 decode_helpers.c File Reference

Functions

- void dump_buffer (void *buffer, __u16 size)
  dump_buffer takes a pointer and dumps a specific number of bytes
- void reverse_array_10 (__u8 bytes[ ])
  This function reverses a 10 byte array.
- struct proto_msg * mem_to_msg (const __u8 *ptr)
  This function is used to decode protobuf messages and message types.
- void free_mem_to_msg (struct proto_msg **msg)
  Frees the Pointer returned by a previous successful call to mem_to_msg.
- struct bgp_ipv4_prefix * read_bgp_prefix (__u8 *ptr)
  Reads a BGP encoded prefix from an NLRI or withdraw message.
- void free_bgp_prefix (struct bgp_ipv4_prefix **prefix)
  Frees the Pointer returned by a previous successful call to read_bgp_prefix.
- __u8 get_var_int (const u_char *data, __u64 *varint)
  this function gets a variable encoded 64 bit integer from a 7 byte encoded string of bytes.

5.3.1 Function Documentation

5.3.1.1 dump_buffer()

```
void dump_buffer (
            void * buffer,
            __u16 size )
```

dump_buffer takes a pointer and dumps a specific number of bytes

This function dumps memory in a format that is importable by wireshark or other programs that can import hex dumps

Parameters

in	*buffer*	A pointer to the memory to be dumped
in	*size*	The number of bytes to dump

5.3.1.2 free_bgp_prefix()

```
void free_bgp_prefix (
            struct bgp_ipv4_prefix ** prefix )
```

Frees the Pointer returned by a previous successful call to read_bgp_prefix.

NOTE: This function should only be called on a pointer previously returned by read_bgp_prefix This function takes a double pointer so that it can free the inner pointer and set it to NULL

Parameters

in	*prefix*	A double pointer with the inner pointer being the previously allocated structure

5.3.1.3 free_mem_to_msg()

```
void free_mem_to_msg (
            struct proto_msg ** msg )
```

Frees the Pointer returned by a previous successful call to mem_to_msg.

NOTE: This function should only be called on a pointer previously returned by mem_to_msg This function takes a double pointer so that it can free the inner pointer and set it to NULL

Parameters

in	*msg*	A double pointer with the inner pointer being the previously allocated structure

5.3.1.4 get_var_int()

```
__u8 get_var_int (
            const u_char * data,
            __u64 * varint )
```

this function gets a variable encoded 64 bit integer from a 7 byte encoded string of bytes.

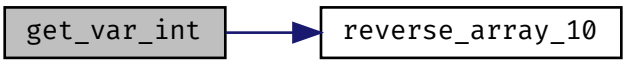
Parameters

in	*data*	A pointer to the bytes we are extracting the variable integer from
in	*varint*	A pointer to where the extracted varint will be stored

Returns

The number of bytes used to encode the extracted varint

Here is the call graph for this function:



5.3.1.5 mem_to_msg()

```
struct proto_msg* mem_to_msg (
            const __u8 * ptr )
```

This function is used to decode protobuf messages and message types.

This function reads a maximum of 10 bytes of memory into an array, terminating when one byte does not have its high order bit set. It then reverses the array, and concatenates the byte array into a single 64 bit integer, using only the low order 7 bits of each byte in the array. The result is then shifted right by 3 to produce the message code and AND'd by 7 to get the high order bytes to find the message type NOTE: It is important that this function returns allocated memory, and the result must be free'd by the calling function to avoid memory leaks.

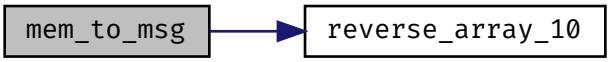
Parameters

in	*ptr*	A pointer to the memory containing the bytes requiring decode

Returns

An allocated proto_msg structure or NULL if allocation of memory fails

Here is the call graph for this function:



5.3.1.6 read_bgp_prefix()

```
struct bgp_ipv4_prefix* read_bgp_prefix (
            __u8 * ptr )
```

Reads a BGP encoded prefix from an NLRI or withdraw message.

See also

bgp_ipv4_prefix

This function decodes compressed ipv4 prefix's as contained in bgp update messages. The first byte in the message represents the CIDR mask (The number of bits in the prefix). If the CIDR is fully divisible by 8, then CIDR/8 bytes are copied into the prefix element. If CIDR is NOT fully divisible by 8 then (CIDR/8)+1 bytes are copied into the prefix element. The next pointer element is set using the input pointer + 1 byte for the CIDR and then X bytes for the prefix itself, where X is either CIDR/8 or (CIDR/8)+1 NOTE: This function returns a pointer that must be free'd by the caller to avoid memory leaks. This function only serves for demonstration purposes because this type of processing would normally be done inline in a BGP update function without the need of additional memory allocation.

Parameters

in	*ptr*	A Pointer to the start of the compressed prefix

5.3.1.7 reverse_array_10()

```
void reverse_array_10 (
```

```
__u8 bytes[] )
```

This function reverses a 10 byte array.

This function is used primarily for handling variable integer decodes VARINT's are something used heavily in gbp file formats (Google protobuffers) NOTE: This function assumes that the input array is at least 10 bytes long and should the input array be less than 10 bytes this may cause unexpected behavior

Parameters

in	*bytes*	An array of bytes to be reversed.

5.4 decode_helpers.h File Reference

Data Structures

- struct bgp_ipv4_prefix

  This structure contains a prefix (in network byte order) and a CIDR for the prefix.
- struct proto_msg

  This structure contains elements used after decoding a protobuf string.

Functions

- void dump_buffer (void *buffer, __u16 size)

  dump_buffer takes a pointer and dumps a specific number of bytes
- void reverse_array_10 (__u8 bytes[])

  This function reverses a 10 byte array.
- struct proto_msg * mem_to_msg (const __u8 *ptr)

  This function is used to decode protobuf messages and message types.
- struct bgp_ipv4_prefix * read_bgp_prefix (__u8 *ptr)

  Reads a BGP encoded prefix from an NLRI or withdraw message.
- void free_mem_to_msg (struct proto_msg **msg)

  Frees the Pointer returned by a previous successful call to mem_to_msg.
- __u8 get_var_int (const u_char *data, __u64 *varint)

  this function gets a variable encoded 64 bit integer from a 7 byte encoded string of bytes.
- void free_bgp_prefix (struct bgp_ipv4_prefix **prefix)

  Frees the Pointer returned by a previous successful call to read_bgp_prefix.

5.4.1 Function Documentation

5.4.1.1 dump_buffer()

```
void dump_buffer (
            void * buffer,
            __u16 size )
```

dump_buffer takes a pointer and dumps a specific number of bytes

This function dumps memory in a format that is importable by wireshark or other programs that can import hex dumps

Parameters

in	*buffer*	A pointer to the memory to be dumped
in	*size*	The number of bytes to dump

5.4.1.2 free_bgp_prefix()

```
void free_bgp_prefix (
            struct bgp_ipv4_prefix ** prefix )
```

Frees the Pointer returned by a previous successful call to read_bgp_prefix.

NOTE: This function should only be called on a pointer previously returned by read_bgp_prefix This function takes a double pointer so that it can free the inner pointer and set it to NULL

Parameters

in	*prefix*	A double pointer with the inner pointer being the previously allocated structure

5.4.1.3 free_mem_to_msg()

```
void free_mem_to_msg (
            struct proto_msg ** msg )
```

Frees the Pointer returned by a previous successful call to mem_to_msg.

NOTE: This function should only be called on a pointer previously returned by mem_to_msg This function takes a double pointer so that it can free the inner pointer and set it to NULL

Parameters

in	*msg*	A double pointer with the inner pointer being the previously allocated structure

5.4.1.4 get_var_int()

```
__u8 get_var_int (
            const u_char * data,
            __u64 * varint )
```

this function gets a variable encoded 64 bit integer from a 7 byte encoded string of bytes.

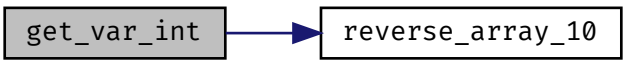
Parameters

in	*data*	A pointer to the bytes we are extracting the variable integer from
in	*varint*	A pointer to where the extracted varint will be stored

Returns

The number of bytes used to encode the extracted varint

Here is the call graph for this function:



5.4.1.5 mem_to_msg()

```
struct proto_msg* mem_to_msg (
            const __u8 * ptr )
```

This function is used to decode protobuf messages and message types.

This function reads a maximum of 10 bytes of memory into an array, terminating when one byte does not have its high order bit set. It then reverses the array, and concatenates the byte array into a single 64 bit integer, using only the low order 7 bits of each byte in the array. The result is then shifted right by 3 to produce the message code and AND'd by 7 to get the high order bytes to find the message type NOTE: It is important that this function returns allocated memory, and the result must be free'd by the calling function to avoid memory leaks.

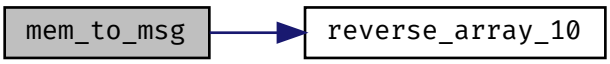
Parameters

in	*ptr*	A pointer to the memory containing the bytes requiring decode

Returns

An allocated proto_msg structure or NULL if allocation of memory fails

Here is the call graph for this function:



5.4.1.6 read_bgp_prefix()

```
struct bgp_ipv4_prefix* read_bgp_prefix (
            __u8 * ptr )
```

Reads a BGP encoded prefix from an NLRI or withdraw message.

See also

bgp_ipv4_prefix

This function decodes compressed ipv4 prefix's as contained in bgp update messages. The first byte in the message represents the CIDR mask (The number of bits in the prefix). If the CIDR is fully divisible by 8, then CIDR/8 bytes are copied into the prefix element. If CIDR is NOT fully divisible by 8 then (CIDR/8)+1 bytes are copied into the prefix element. The next pointer element is set using the input pointer + 1 byte for the CIDR and then X bytes for the prefix itself, where X is either CIDR/8 or (CIDR/8)+1 NOTE: This function returns a pointer that must be free'd by the caller to avoid memory leaks. This function only serves for demonstration purposes because this type of processing would normally be done inline in a BGP update function without the need of additional memory allocation.

Parameters

in	*ptr*	A Pointer to the start of the compressed prefix

5.4.1.7 reverse_array_10()

```
void reverse_array_10 (
            __u8 bytes[] )
```

This function reverses a 10 byte array.

This function is used primarily for handling variable integer decodes VARINT's are something used heavily in gbp file formats (Google protobuffers) NOTE: This function assumes that the input array is at least 10 bytes long and should the input array be less than 10 bytes this may cause unexpected behavior

Parameters

in	*bytes*	An array of bytes to be reversed.

5.5 linked_list.c File Reference

Functions

- struct route4tree ∗ init_tree4 (void)

  Initialize a binary tree head entry.
- struct route4tree ∗ insert_tree4 (struct route4tree ∗tree, __u32 addr, __u8 cidr, void ∗data, callback cb)

  This inserts an entry into a binary tree.
- struct route4tree ∗ lookup_lpm (struct route4tree ∗tree, __u32 addr, __u8 cidr)

  This does a longest prefix match against the binary tree.
- struct route4tree ∗ lookup_exact (struct route4tree ∗tree, __u32 addr, __u8 cidr)

  This does an exact match lookup against the tree.
- void remove_node (struct route4tree ∗tree, __u32 address, __u8 cidr, void ∗data, callback_remove cb)

  This function removes a node from the tree.
- void free_tree4 (struct route4tree ∗∗tree)

  Frees an entire 32-bit binary tree using post-order traversal.

5.5.1 Function Documentation

5.5.1.1 free_tree4()

```
void free_tree4 (
            struct route4tree ** tree )
```

Frees an entire 32-bit binary tree using post-order traversal.

This function traverses the entire tree starting from the given node and frees all dynamically allocated memory for:

- child nodes (`set` and `unset` pointers)
- the `data` pointer stored at each node
- the node itself

The traversal is performed in a post-order manner (children first, then parent) to ensure no node is freed before its children, avoiding use-after-free errors. Parent pointers are used to explicitly walk back up the tree without recursion.

After calling this function on the root of a tree, all memory associated with that tree is released. The caller must not use any pointer to this tree or its nodes after this function returns.

Parameters

in	*tree*	A double pointer to the root of a 32-bit binary tree (`struct route4tree`) to be freed. If the inner pointer is NULL, the function does nothing.

5.5.1.2 init_tree4()

```
struct route4tree* init_tree4 (
            void  )
```

Initialize a binary tree head entry.

Returns

A pointer to the head of a binary tree

5.5.1.3 insert_tree4()

```
struct route4tree* insert_tree4 (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr,
            void * data,
            callback cb )
```

This inserts an entry into a binary tree.

Data is a void pointer making this generic and able to store any data in the tree. It should be noted that the tree must be initialized before first insertion using init_tree4();

Parameters

in	*tree*	An initialized binary tree
in	*addr*	A 32 bit address to be inserted into the tree
in	*cidr*	An integer representing how many bits of the address to insert into the tree
in	*data*	A void pointer to the data to insert into the tree
in	*cb*	A Function pointer used for inserting data - if NULL the tree data pointer will simply point to the data parameter

Returns

A route4tree structure at the inserted position in the tree

5.5.1.4 lookup_exact()

```
struct route4tree* lookup_exact (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr )
```

This does an exact match lookup against the tree.

This function will return the tree structure at the exact match point if it can transverse the tree to the depths represented by cidr and if data is present at the end of the transversal

Parameters

in	*tree*	The head of an initialized binary tree
in	*addr*	An address to lookup in the tree
in	*cidr*	The depth to transverse to in the tree

Returns

A pointer to the tree entry if matched, or NULL

5.5.1.5 lookup_lpm()

```
struct route4tree* lookup_lpm (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr )
```

This does a longest prefix match against the binary tree.

This function will return at the deepest point in the tree that is matched and has data. It is useful in the data pointer to store the actual prefix and cidr so that when this returns you can know exactly what has been matched.

Parameters

in	*tree*	The head of an initialized binary tree
in	*addr*	A 32 bit address to be searched for in the tree
in	*cidr*	The maximum number of bits to transverse to in the tree

Returns

A tree entry at the deepest point possible in the transversal

5.5.1.6 remove_node()

```
void remove_node (
            struct route4tree * tree,
            __u32 address,
            __u8 cidr,
            void * data,
            callback_remove cb )
```

This function removes a node from the tree.

This function will remove a node from the tree, and all empty parent nodes. If a parent node still has children or data the function will cease the reverse transversal

Parameters

in	*tree*	The head of an initialized binary tree
in	*address*	The address to lookup in the tree - This should be in big endian order
in	*cidr*	The depth to transverse before we start back tracing and freeing
in	*data*	If this is set AND callback is set, the callback is used with the node data and the supplied data to determine removal
in	*cb*	If set call the callback removal function with the node data pointer and the supplied data pointer

5.6 linked_list.h File Reference

Data Structures

- struct route4tree

  The route4tree structure is used in the creation of a 32 bit binary tree.

Typedefs

- typedef int() callback(struct route4tree ∗, void ∗)
- typedef int() callback_remove(struct route4tree ∗, void ∗, void ∗)

Functions

- struct route4tree ∗ init_tree4 (void)

  Initialize a binary tree head entry.
- struct route4tree ∗ insert_tree4 (struct route4tree ∗tree, __u32 addr, __u8 cidr, void ∗data, callback cb)

  This inserts an entry into a binary tree.
- struct route4tree ∗ lookup_lpm (struct route4tree ∗tree, __u32 addr, __u8 cidr)

  This does a longest prefix match against the binary tree.
- struct route4tree ∗ lookup_exact (struct route4tree ∗tree, __u32 addr, __u8 cidr)

  This does an exact match lookup against the tree.
- void remove_node (struct route4tree ∗tree, __u32 address, __u8 cidr, void ∗data, callback_remove cb)

  This function removes a node from the tree.
- void free_tree4 (struct route4tree ∗∗tree)

  Frees an entire 32-bit binary tree using post-order traversal.

5.6.1 Typedef Documentation

5.6.1.1 callback

```
typedef int() callback(struct route4tree *, void *)
```

5.6.1.2 callback_remove

```
typedef int() callback_remove(struct route4tree *, void *, void *)
```

5.6.2 Function Documentation

5.6.2.1 free_tree4()

```
void free_tree4 (
            struct route4tree ** tree )
```

Frees an entire 32-bit binary tree using post-order traversal.

This function traverses the entire tree starting from the given node and frees all dynamically allocated memory for:

- child nodes (`set` and `unset` pointers)
- the `data` pointer stored at each node
- the node itself

The traversal is performed in a post-order manner (children first, then parent) to ensure no node is freed before its children, avoiding use-after-free errors. Parent pointers are used to explicitly walk back up the tree without recursion.

After calling this function on the root of a tree, all memory associated with that tree is released. The caller must not use any pointer to this tree or its nodes after this function returns.

Parameters

`in`	*tree*	A double pointer to the root of a 32-bit binary tree (`struct route4tree`) to be freed. If the inner pointer is NULL, the function does nothing.

5.6.2.2 init_tree4()

```
struct route4tree* init_tree4 (
            void  )
```

Initialize a binary tree head entry.

Returns

A pointer to the head of a binary tree

5.6.2.3 insert_tree4()

```
struct route4tree* insert_tree4 (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr,
            void * data,
            callback cb )
```

This inserts an entry into a binary tree.

Data is a void pointer making this generic and able to store any data in the tree. It should be noted that the tree must be initialized before first insertion using init_tree4();

Parameters

in	*tree*	An initialized binary tree
in	*addr*	A 32 bit address to be inserted into the tree
in	*cidr*	An integer representing how many bits of the address to insert into the tree
in	*data*	A void pointer to the data to insert into the tree
in	*cb*	A Function pointer used for inserting data - if NULL the tree data pointer will simply point to the data parameter

Returns

A route4tree structure at the inserted position in the tree

5.6.2.4 lookup_exact()

```
struct route4tree* lookup_exact (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr )
```

This does an exact match lookup against the tree.

This function will return the tree structure at the exact match point if it can transverse the tree to the depths represented by cidr and if data is present at the end of the transversal

Parameters

in	*tree*	The head of an initialized binary tree
in	*addr*	An address to lookup in the tree
in	*cidr*	The depth to transverse to in the tree

Returns

A pointer to the tree entry if matched, or NULL

5.6.2.5 lookup_lpm()

```
struct route4tree* lookup_lpm (
            struct route4tree * tree,
            __u32 addr,
            __u8 cidr )
```

This does a longest prefix match against the binary tree.

This function will return at the deepest point in the tree that is matched and has data. It is useful in the data pointer to store the actual prefix and cidr so that when this returns you can know exactly what has been matched.

Parameters

in	*tree*	The head of an initialized binary tree
in	*addr*	A 32 bit address to be searched for in the tree
in	*cidr*	The maximum number of bits to transverse to in the tree

Returns

A tree entry at the deepest point possible in the transversal

5.6.2.6 remove_node()

```
void remove_node (
            struct route4tree * tree,
            __u32 address,
            __u8 cidr,
            void * data,
            callback_remove cb )
```

This function removes a node from the tree.

This function will remove a node from the tree, and all empty parent nodes. If a parent node still has children or data the function will cease the reverse transversal

Parameters

in	*tree*	The head of an initialized binary tree
in	*address*	The address to lookup in the tree - This should be in big endian order
in	*cidr*	The depth to transverse before we start back tracing and freeing
in	*data*	If this is set AND callback is set, the callback is used with the node data and the supplied data to determine removal
in	*cb*	If set call the callback removal function with the node data pointer and the supplied data pointer

5.7 README.md File Reference

5.8 tests/test_linked_list.c File Reference

Data Structures

- struct data

Functions

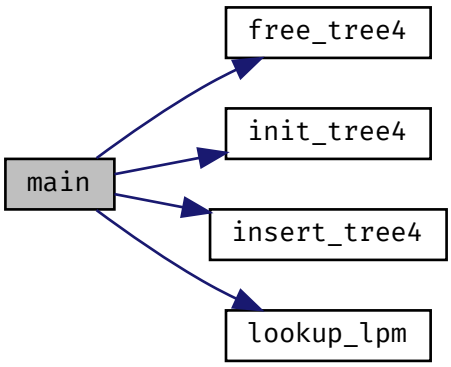
- int main (void)

5.8.1 Function Documentation

5.8.1.1 main()

```
int main (
            void  )
```

Here is the call graph for this function:



Index