My Project

Generated by Doxygen 1.15.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

BIT_READER . . . . **??**
BitWriter . . . . **??**
CanonicalCode . . . . **??**
DebugmallocData . . . . **??**
DebugmallocEntry . . . . **??**
DISTANCE_CODE . . . . **??**
HUFFMAN_CODE . . . . **??**
HuffmanEntry . . . . **??**
HuffmanTree . . . . **??**
LENGTH_CODE . . . . **??**
LZ77_buffer
Structure to manage a dynamic array (growing buffer) of LZ77 tokens . . . . **??**
LZ77_compressed
The fundamental LZ77 token structure . . . . **??**
MinHeap . . . . **??**
Node . . . . **??**
Status . . . . **??**

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

bitreader.c . . . . **??**
bitreader.h . . . . **??**
bitwriter.c . . . . **??**
bitwriter.h . . . . **??**
compress.c . . . . **??**
compress.h . . . . **??**
CRC_CHECKSUM.c . . . . **??**
CRC_CHECKSUM.h . . . . **??**
debugmalloc.h . . . . **??**
decompress.c . . . . **??**
decompress.h . . . . **??**
distance.c . . . . **??**
distance.h . . . . **??**
HUFFMAN_TABLE.c . . . . **??**
HUFFMAN_TABLE.h . . . . **??**
length.c . . . . **??**
length.h . . . . **??**
LZ77.c . . . . **??**
LZ77.h . . . . **??**
main.c . . . . **??**
node.c . . . . **??**
node.h . . . . **??**
status.h . . . . **??**

Chapter 3

Data Structure Documentation

3.1 BIT_READER Struct Reference

```
#include <bitreader.h>
```

Data Fields

- FILE ∗ **file**
- uint8_t ∗ **buffer**
- uint8_t **byte**
- uint8_t **currentPosition**
- size_t **bufferSize**
- uint16_t **index**
- char ∗ **fileName**

3.1.1 Field Documentation

3.1.1.1 buffer

```
uint8_t* buffer
```

3.1.1.2 bufferSize

```
size_t bufferSize
```

3.1.1.3 byte

```
uint8_t byte
```

3.1.1.4 currentPosition

```
uint8_t currentPosition
```

3.1.1.5 file

```
FILE* file
```

3.1.1.6 fileName

```
char* fileName
```

3.1.1.7 index

```
uint16_t index
```

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

- **bitreader.h**

3.2 BitWriter Struct Reference

```
#include <bitwriter.h>
```

Data Fields

- FILE * **file**
- uint8_t * **buffer**
- uint8_t **byte**
- uint8_t **currentPosition**
- size_t **bufferSize**
- uint16_t **index**
- char * **fileName**

3.2.1 Field Documentation

3.2.1.1 buffer

```
uint8_t* buffer
```

3.2.1.2 bufferSize

```
size_t bufferSize
```

3.2.1.3 byte

```
uint8_t byte
```

3.2.1.4 currentPosition

```
uint8_t currentPosition
```

3.2.1.5 file

```
FILE* file
```

3.2.1.6 fileName

```
char* fileName
```

3.2.1.7 index

```
uint16_t index
```

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

- **bitwriter.h**

3.3 CanonicalCode Struct Reference

```
#include <HUFFMAN_TABLE.h>
```

Data Fields

- uint16_t **code**
- uint8_t **length**

3.3.1 Field Documentation

3.3.1.1 code

```
uint16_t code
```

3.3.1.2 length

```
uint8_t length
```

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

- **HUFFMAN_TABLE.h**

3.4 DebugmallocData Struct Reference

```
#include <debugmalloc.h>
```

Collaboration diagram for DebugmallocData:

Data Fields

- char **logfile** [256]
- long **max_block_size**
- long **alloc_count**
- long long **alloc_bytes**
- long **all_alloc_count**
- long long **all_alloc_bytes**
- **DebugmallocEntry head** [**debugmalloc_tablesize**]
- **DebugmallocEntry tail** [**debugmalloc_tablesize**]

3.4.1 Field Documentation

3.4.1.1 all_alloc_bytes

```
long long all_alloc_bytes
```

3.4.1.2 all_alloc_count

```
long all_alloc_count
```

3.4.1.3 alloc_bytes

```
long long alloc_bytes
```

3.4.1.4 alloc_count

```
long alloc_count
```

3.4.1.5 head

```
DebugmallocEntry head[ debugmalloc_tablesize]
```

3.4.1.6 logfile

```
char logfile[256]
```

3.4.1.7 max_block_size

```
long max_block_size
```

3.4.1.8 tail

```
DebugmallocEntry tail[ debugmalloc_tablesize]
```

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

- **debugmalloc.h**

3.5 DebugmallocEntry Struct Reference

```
#include <debugmalloc.h>
```

Collaboration diagram for DebugmallocEntry:

Data Fields

- void ∗ **real_mem**
- void ∗ **user_mem**
- size_t **size**
- char **file** [64]
- unsigned **line**
- char **func** [32]
- char **expr** [128]
- struct **DebugmallocEntry** ∗ **prev**
- struct **DebugmallocEntry** ∗ **next**

3.5.1 Field Documentation

3.5.1.1 expr

```
char expr[128]
```

3.5.1.2 file

```
char file[64]
```

3.5.1.3 func

```
char func[32]
```

3.5.1.4 line

```
unsigned line
```

3.5.1.5 next

```
struct DebugmallocEntry * next
```

3.5.1.6 prev

```
struct DebugmallocEntry* prev
```

3.5.1.7 real_mem

```
void* real_mem
```

3.5.1.8 size

```
size_t size
```

3.5.1.9 user_mem

```
void* user_mem
```

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

- **debugmalloc.h**

3.6 DISTANCE_CODE Struct Reference

```
#include <distance.h>
```

Data Fields

- unsigned short **usSymbolID**
- int **iExtraBits**
- int **iExtraValue**

3.6.1 Field Documentation

3.6.1.1 iExtraBits

```
int iExtraBits
```

3.6.1.2 iExtraValue

```
int iExtraValue
```

3.6.1.3 usSymbolID

```
unsigned short usSymbolID
```

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

- **distance.h**

3.7 HUFFMAN_CODE Struct Reference

```
#include <HUFFMAN_TABLE.h>
```

Data Fields

- uint16_t **code**
- uint8_t **length**

3.7.1 Field Documentation

3.7.1.1 code

```
uint16_t code
```

3.7.1.2 length

```
uint8_t length
```

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

- **HUFFMAN_TABLE.h**

3.8 HuffmanEntry Struct Reference

```
#include <HUFFMAN_TABLE.h>
```

Data Fields

- uint16_t **symbol**
- uint8_t **bits**

3.8.1 Field Documentation

3.8.1.1 bits

```
uint8_t bits
```

3.8.1.2 symbol

```
uint16_t symbol
```

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

- **HUFFMAN_TABLE.h**

3.9 HuffmanTree Struct Reference

```
#include <HUFFMAN_TABLE.h>
```

Collaboration diagram for HuffmanTree:

Data Fields

- **HuffmanEntry lookup_table** [1<< **FAST_BITS**]
- **CanonicalCode codes_list** [**MAX_CODE_SYMBOLS**]
- uint16_t **total_symbols**
- uint8_t **max_length**

3.9.1 Field Documentation

3.9.1.1 codes_list

```
CanonicalCode codes_list[ MAX_CODE_SYMBOLS]
```

3.9.1.2 lookup_table

```
HuffmanEntry lookup_table[1<<  FAST_BITS]
```

3.9.1.3 max_length

```
uint8_t max_length
```

3.9.1.4 total_symbols

```
uint16_t total_symbols
```

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

- **HUFFMAN_TABLE.h**

3.10 LENGTH_CODE Struct Reference

```
#include <length.h>
```

Data Fields

- unsigned short **usSymbolID**
- int **iExtraBits**
- int **iExtraValue**

3.10.1 Field Documentation

3.10.1.1 iExtraBits

```
int iExtraBits
```

3.10.1.2 iExtraValue

```
int iExtraValue
```

3.10.1.3 usSymbolID

```
unsigned short usSymbolID
```

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

- **length.h**

3.11 LZ77_buffer Struct Reference

Structure to manage a dynamic array (growing buffer) of LZ77 tokens.

```
#include <LZ77.h>
```

Collaboration diagram for LZ77_buffer:

Data Fields

- **LZ77_compressed** $*$ **tokens**
  Pointer to the start of the dynamic array of tokens.
- size_t **size**
  The current number of tokens stored (using size_t for large files).
- size_t **capacity**
  The total number of tokens the buffer can hold.

3.11.1 Detailed Description

Structure to manage a dynamic array (growing buffer) of LZ77 tokens.

- Stores the tokens directly in a contiguous array (LZ77_compressed$*$), avoiding per-token allocations for better performance and cache utilization.

3.11.2 Field Documentation

3.11.2.1 capacity

```
size_t capacity
```

The total number of tokens the buffer can hold.

3.11.2.2 size

```
size_t size
```

The current number of tokens stored (using size_t for large files).

3.11.2.3 tokens

```
LZ77_compressed* tokens
```

Pointer to the start of the dynamic array of tokens.

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

- **LZ77.h**

3.12 LZ77_compressed Struct Reference

The fundamental LZ77 token structure.

```
#include <LZ77.h>
```

Data Fields

- **LZ77_encoded_type type**
- union {
    uint8_t **literal**
    struct {
      uint16_t **distance**
      uint16_t **length**
    } **match**
  } **data**

3.12.1 Detailed Description

The fundamental LZ77 token structure.

- Uses a struct for the match data to correctly store both distance and length. Uses a union to ensure the token only takes the size of the largest data type (the match struct).

3.12.2 Field Documentation

3.12.2.1 [union]

```
union { ... } data
```

3.12.2.2 distance

```
uint16_t distance
```

3.12.2.3 length

```
uint16_t length
```

3.12.2.4 literal

```
uint8_t literal
```

3.12.2.5 [struct]

```
struct { ...  } match
```

3.12.2.6 type

```
LZ77_encoded_type type
```

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

- **LZ77.h**

3.13 MinHeap Struct Reference

```
#include <node.h>
```

Collaboration diagram for MinHeap:

Data Fields

- int **iSize**
- int **iCapacity**
- **Node** ∗∗ **ppnArray**

3.13.1 Field Documentation

3.13.1.1 iCapacity

```
int iCapacity
```

3.13.1.2 iSize

```
int iSize
```

3.13.1.3 ppnArray

```
Node** ppnArray
```

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

- **node.h**

3.14 Node Struct Reference

```
#include <node.h>
```

Collaboration diagram for Node:

Data Fields

- int **iFrequency**
- unsigned short **usSymbol**
- struct **Node** ∗ **pnLeft**
- struct **Node** ∗ **pnRight**

3.14.1 Field Documentation

3.14.1.1 iFrequency

```
int iFrequency
```

3.14.1.2 pnLeft

```
struct Node* pnLeft
```

3.14.1.3 pnRight

```
struct Node* pnRight
```

3.14.1.4 usSymbol

```
unsigned short usSymbol
```

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

- **node.h**

3.15 Status Struct Reference

```
#include <status.h>
```

Data Fields

- **StatusCode code**
- char * **message**

3.15.1 Field Documentation

3.15.1.1 code

```
StatusCode code
```

3.15.1.2 message

```
char* message
```

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

- **status.h**

Chapter 4

File Documentation

4.1 bitreader.c File Reference

```
#include "bitreader.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "debugmalloc.h"
```

Include dependency graph for bitreader.c:

Macros

- #define **BUFFER_SIZE** 4096
- #define **GZIP_ID1** 0x1f
- #define **GZIP_ID2** 0x8b
- #define **GZIP_CM_DEFLATE** 0x08
- #define **FTEXT** 0x01
- #define **FHCRC** 0x02
- #define **FEXTRA** 0x04
- #define **FNAME** 0x08
- #define **FCOMMENT** 0x10

Functions

- uint32_t **read_bits** (**BIT_READER** ∗reader, int numBits)

  Processes the 10-byte GZIP header and any optional fields.
- bool **process_gzip_header** (**BIT_READER** ∗reader)
- **BIT_READER** ∗ **init_bit_reader** (const char ∗filePath)
- int **read_bit** (**BIT_READER** ∗reader)
- void **close_bit_reader** (**BIT_READER** ∗reader)
- uint16_t **peek_bits** (**BIT_READER** ∗reader, uint8_t n)

4.1.1 Macro Definition Documentation

4.1.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 4096
```

4.1.1.2 FCOMMENT

```
#define FCOMMENT 0x10
```

4.1.1.3 FEXTRA

```
#define FEXTRA 0x04
```

4.1.1.4 FHCRC

```
#define FHCRC 0x02
```

4.1.1.5 FNAME

```
#define FNAME 0x08
```

4.1.1.6 FTEXT

```
#define FTEXT 0x01
```

4.1.1.7 GZIP_CM_DEFLATE

```
#define GZIP_CM_DEFLATE 0x08
```

4.1.1.8 GZIP_ID1

```
#define GZIP_ID1 0x1f
```

4.1.1.9 GZIP_ID2

```
#define GZIP_ID2 0x8b
```

4.1.2 Function Documentation

4.1.2.1 close_bit_reader()

```
void close_bit_reader (
              BIT_READER * reader)
```

Closes the file handle and performs cleanup.

Parameters

reader	Pointer to the **BIT_READER** (p. **??**) structure.

4.1.2.2 init_bit_reader()

```
BIT_READER * init_bit_reader (
            const char * filePath)
```

Initializes the **BIT_READER** (p. **??**) structure. Opens the file and resets the bit-reading state.

Parameters

filePath	Path to the input file.

Returns

0 on success, -1 on failure (e.g., file not found).

4.1.2.3 peek_bits()

```
uint16_t peek_bits (
              BIT_READER * reader,
             uint8_t n)  [extern]
```

4.1.2.4 process_gzip_header()

```
bool process_gzip_header (
              BIT_READER * reader)
```

4.1.2.5 read_bit()

```
int read_bit (
              BIT_READER * reader)
```

Reads a single bit from the stream. Handles reading new bytes from the file when the current byte is exhausted.

Parameters

reader	Pointer to the initialized **BIT_READER** (p. **??**).

Returns

The bit value (0 or 1), or -1 if the end of file is reached unexpectedly.

4.1.2.6 read_bits()

```
uint32_t read_bits (
            BIT_READER * reader,
            int numBits)
```

Processes the 10-byte GZIP header and any optional fields.

- **Parameters**

reader	Pointer to the initialized **BIT_READER** (p. **??**).

Returns

true if the header is valid and optional fields were processed (or skipped).
false if the file is not a valid GZIP stream or required features are unsupported.

4.2 bitreader.h File Reference

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

Include dependency graph for bitreader.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **BIT_READER**

Functions

- **BIT_READER** ∗ **init_bit_reader** (const char ∗filePath)
- int **read_bit** (**BIT_READER** ∗reader)
- uint32_t **read_bits** (**BIT_READER** ∗reader, int numBits)

  Processes the 10-byte GZIP header and any optional fields.
- void **close_bit_reader** (**BIT_READER** ∗reader)
- bool **process_gzip_header** (**BIT_READER** ∗reader)
- uint16_t **peek_bits** (**BIT_READER** ∗reader, uint8_t n)

4.2.1 Function Documentation

4.2.1.1 close_bit_reader()

```
void close_bit_reader (
              BIT_READER * reader)
```

Closes the file handle and performs cleanup.

Parameters

reader	Pointer to the **BIT_READER** (p. **??**) structure.

4.2.1.2 init_bit_reader()

```
BIT_READER * init_bit_reader (
             const char * filePath)
```

Initializes the **BIT_READER** (p. **??**) structure. Opens the file and resets the bit-reading state.

Parameters

filePath	Path to the input file.

Returns

0 on success, -1 on failure (e.g., file not found).

4.2.1.3 peek_bits()

```
uint16_t peek_bits (
             BIT_READER * reader,
             uint8_t n)  [extern]
```

4.2.1.4 process_gzip_header()

```
bool process_gzip_header (
             BIT_READER * reader)
```

4.2.1.5 read_bit()

```
int read_bit (
             BIT_READER * reader)
```

Reads a single bit from the stream. Handles reading new bytes from the file when the current byte is exhausted.

Parameters

reader	Pointer to the initialized **BIT_READER** (p. **??**).

Returns

The bit value (0 or 1), or -1 if the end of file is reached unexpectedly.

4.2.1.6 read_bits()

```
uint32_t read_bits (
             BIT_READER * reader,
             int numBits)
```

Processes the 10-byte GZIP header and any optional fields.

Reads a specified number of bits from the stream.

Parameters

reader	Pointer to the initialized **BIT_READER** (p. **??**).
numBits	The number of bits to read (must be <= 32).

Returns

The unsigned integer value represented by the bits, or 0xFFFFFFFF on error.

- **Parameters**

reader	Pointer to the initialized **BIT_READER** (p. **??**).

  Returns

  true if the header is valid and optional fields were processed (or skipped).
  false if the file is not a valid GZIP stream or required features are unsupported.

4.3 bitreader.h

Go to the documentation of this file.

```
//
// Created by Attila on 11/24/2025.
//

#ifndef DEFLATE_BITREADER_H
#define DEFLATE_BITREADER_H
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

typedef struct {
    FILE *file;
    uint8_t* buffer;
    uint8_t byte;
    uint8_t currentPosition;
    size_t bufferSize;
    uint16_t index;
    char* fileName;
} BIT_READER;

BIT_READER* init_bit_reader(const char *filePath);

int read_bit(BIT_READER *reader);

uint32_t read_bits(BIT_READER *reader, int numBits);

void close_bit_reader(BIT_READER *reader);

//void createFile(BIT_READER* bw, char* fileName, char* extension);

bool process_gzip_header(BIT_READER *reader);

extern uint16_t peek_bits(BIT_READER* reader, uint8_t n);
#endif //DEFLATE_BITREADER_H
```

4.4 bitwriter.c File Reference

```
#include "bitwriter.h"
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Include dependency graph for bitwriter.c:

Macros

- #define **BUFFER_SIZE** 4096
- #define **MAGIC_NUMER** 0x8B1F
- #define **COMPRESSION_METHOD** 0x08
- #define **FLAG** 0b00000000
- #define **XFL** 0x00
- #define **OS** 0x03

Functions

- size_t **flushBitWriterBuffer** (**BitWriter** ∗bw)
- **BitWriter** ∗ **initBitWriter** (void)
- void **addData** (**BitWriter** ∗bw, uint32_t value, uint8_t bitLength)
- void **flush_bitstream_writer** (**BitWriter** ∗bw)
- void **addBytesFromMSB** (**BitWriter** ∗bw, uint32_t value, uint8_t bytes)
- void **addBytesFromMSB2** (**BitWriter** ∗bw, uint32_t value, uint8_t bytes)
- void **createFile** (**BitWriter** ∗bw, char ∗fileName, char ∗extension)
- void **freeBitWriter** (**BitWriter** ∗bw)

4.4.1 Macro Definition Documentation

4.4.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 4096
```

4.4.1.2 COMPRESSION_METHOD

```
#define COMPRESSION_METHOD 0x08
```

4.4.1.3 FLAG

```
#define FLAG 0b00000000
```

4.4.1.4 MAGIC_NUMER

```
#define MAGIC_NUMER 0x8B1F
```

4.4.1.5 OS

```
#define OS 0x03
```

4.4.1.6 XFL

```
#define XFL 0x00
```

4.4.2 Function Documentation

4.4.2.1 addBytesFromMSB()

```
void addBytesFromMSB (
             BitWriter * bw,
             uint32_t value,
             uint8_t bytes)  [extern]
```

4.4.2.2 addBytesFromMSB2()

```
void addBytesFromMSB2 (
             BitWriter * bw,
             uint32_t value,
             uint8_t bytes)  [extern]
```

4.4.2.3 addData()

```
void addData (
             BitWriter * bw,
             uint32_t value,
             uint8_t bitLength)  [extern]
```

4.4.2.4 createFile()

```
void createFile (
             BitWriter * bw,
             char * fileName,
             char * extension)  [extern]
```

4.4.2.5 flush_bitstream_writer()

```
void flush_bitstream_writer (
             BitWriter * bw)  [extern]
```

4.4.2.6 flushBitWriterBuffer()

```
size_t flushBitWriterBuffer (
             BitWriter * bw)  [extern]
```

4.4.2.7 freeBitWriter()

```
void freeBitWriter (
             BitWriter * bw)  [extern]
```

4.4.2.8 initBitWriter()

```
 BitWriter * initBitWriter (
            void )  [extern]
```

4.5 bitwriter.h File Reference

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

Include dependency graph for bitwriter.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **BitWriter**

Functions

- **BitWriter** ∗ **initBitWriter** (void)
- void **addData** (**BitWriter** ∗bw, uint32_t value, uint8_t bitLength)
- void **createFile** (**BitWriter** ∗bw, char ∗fileName, char ∗extension)
- void **freeBitWriter** (**BitWriter** ∗bw)
- void **addBytesFromMSB** (**BitWriter** ∗bw, uint32_t value, uint8_t bytes)
- void **addBytesFromMSB2** (**BitWriter** ∗bw, uint32_t value, uint8_t bytes)
- void **flush_bitstream_writer** (**BitWriter** ∗bw)

4.5.1 Function Documentation

4.5.1.1 addBytesFromMSB()

```
void addBytesFromMSB (
             BitWriter * bw,
            uint32_t value,
            uint8_t bytes)  [extern]
```

4.5.1.2 addBytesFromMSB2()

```
void addBytesFromMSB2 (
             BitWriter * bw,
            uint32_t value,
            uint8_t bytes)  [extern]
```

4.5.1.3 addData()

```
void addData (
             BitWriter * bw,
            uint32_t value,
            uint8_t bitLength)
```

4.5.1.4 createFile()

```
void createFile (
              BitWriter * bw,
             char * fileName,
             char * extension)
```

4.5.1.5 flush_bitstream_writer()

```
void flush_bitstream_writer (
              BitWriter * bw)  [extern]
```

4.5.1.6 freeBitWriter()

```
void freeBitWriter (
              BitWriter * bw)
```

4.5.1.7 initBitWriter()

```
BitWriter * initBitWriter (
             void )
```

4.6 bitwriter.h

Go to the documentation of this file.

```
//
// Created by Attila on 11/19/2025.
//

#ifndef DEFLATE_BITWRITER_H
#define DEFLATE_BITWRITER_H

#include <stdint.h>
#include <stdio.h>

typedef struct {
    FILE *file;
    uint8_t* buffer;
    uint8_t byte;
    uint8_t currentPosition;
    size_t bufferSize;
    uint16_t index;
    char* fileName;
} BitWriter;

BitWriter* initBitWriter(void);

void addData(BitWriter* bw, uint32_t value, uint8_t bitLength);

void createFile(BitWriter* bw, char* fileName, char* extension);

void freeBitWriter(BitWriter* bw);

extern void addBytesFromMSB(BitWriter* bw, uint32_t value, uint8_t bytes);

extern void addBytesFromMSB2(BitWriter* bw, uint32_t value, uint8_t bytes);

extern void flush_bitstream_writer(BitWriter* bw);

#endif //DEFLATE_BITWRITER_H
```

4.7 compress.c File Reference

```
#include "compress.h"
#include "debugmalloc.h"
#include <stdio.h>
#include <string.h>
#include "bitwriter.h"
#include "CRC_CHECKSUM.h"
#include "distance.h"
#include "HUFFMAN_TABLE.h"
#include "length.h"
#include "LZ77.h"
#include "node.h"
#include "status.h"
```

Include dependency graph for compress.c:

Macros

- #define **HASH_BITS** 15
- #define **HASH_SHIFT** 5
- #define **HASH_MASK** 0x7FFF
- #define **HASH_SIZE** (1 << **HASH_BITS**)
- #define **WINDOW_SIZE** 32768
- #define **BUFFER_SIZE** (**WINDOW_SIZE** ∗ 2)
- #define **EMPTY_INDEX** 0xFFFF
- #define **LITERAL_LENGTH_SIZE** 286
- #define **END_OF_BLOCK** 256
- #define **DISTANCE_CODE_SIZE** 30
- #define **CODE_LENGTH_FREQUENCIES** 19
- #define **BYTE** uint8_t

Functions

- FILE ∗ **ffOpenFile** (const char ∗filename)

  Opens a file in rb (read binary) mode.
- void **compressData** (const unsigned char ∗ucpBuffer, const size_t bytesRead, uint16_t ∗hash_table, **LZ77_buffer** ∗output_ucpBuffer)

  Compress Data.
- void **writeGzipTrailer** (**BitWriter** ∗bw, uint32_t crc32_checksum, uint32_t total_uncompressed_size)
- void **processBlock** (**BitWriter** ∗bw, uint16_t ∗LLFrequency, uint16_t ∗distanceCodeFrequency, const **LZ77_buffer** ∗output_ucpBuffer, const bool lastBlock)
- **Status** **compress** (char ∗filename)

4.7.1 Macro Definition Documentation

4.7.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE ( WINDOW_SIZE * 2)
```

4.7.1.2 BYTE

```
#define BYTE uint8_t
```

4.7.1.3 CODE_LENGTH_FREQUENCIES

```
#define CODE_LENGTH_FREQUENCIES 19
```

4.7.1.4 DISTANCE_CODE_SIZE

```
#define DISTANCE_CODE_SIZE 30
```

4.7.1.5 EMPTY_INDEX

```
#define EMPTY_INDEX 0xFFFF
```

4.7.1.6 END_OF_BLOCK

```
#define END_OF_BLOCK 256
```

4.7.1.7 HASH_BITS

```
#define HASH_BITS 15
```

4.7.1.8 HASH_MASK

```
#define HASH_MASK 0x7FFF
```

4.7.1.9 HASH_SHIFT

```
#define HASH_SHIFT 5
```

4.7.1.10 HASH_SIZE

```
#define HASH_SIZE (1 << HASH_BITS)
```

4.7.1.11 LITERAL_LENGTH_SIZE

```
#define LITERAL_LENGTH_SIZE 286
```

4.7.1.12 WINDOW_SIZE

```
#define WINDOW_SIZE 32768
```

4.7.2 Function Documentation

4.7.2.1 compress()

```
 Status compress (
             char * filename)  [extern]
```

4.7.2.2 compressData()

```
void compressData (
             const unsigned char * ucpBuffer,
             const size_t bytesRead,
             uint16_t * hash_table,
              LZ77_buffer * output_ucpBuffer)  [extern]
```

Compress Data.

This function takes in a BUFFER SIZED buffer containing BYTES from a file, and fills up an **LZ77_buffer** (p. **??**) containing match/literal distance/length codes which will be used later in the processBlock function.

Parameters

ucpBuffer	The start of the buffer (pointer).
bytesRead	The bytes processed in this function.
hash_table	The hash lookup table for matches.
output_ucpBuffer	The **LZ77_buffer** (p. **??**) containing the matches/literals.

Returns

void

4.7.2.3 ffOpenFile()

```
FILE * ffOpenFile (
             const char * filename)  [extern]
```

Opens a file in rb (read binary) mode.

Parameters

filename	The name of the file. (probably with absolute path)

Returns

FILE∗ or NULL

4.7.2.4 processBlock()

```
void processBlock (
              BitWriter * bw,
             uint16_t * LLFrequency,
             uint16_t * distanceCodeFrequency,
             const  LZ77_buffer * output_ucpBuffer,
             const bool lastBlock)  [extern]
```

4.7.2.5 writeGzipTrailer()

```
void writeGzipTrailer (
              BitWriter * bw,
             uint32_t crc32_checksum,
             uint32_t total_uncompressed_size)  [extern]
```

4.8 compress.h File Reference

```
#include <stddef.h>
#include "bitwriter.h"
#include "status.h"
```

Include dependency graph for compress.h: This graph shows which files directly or indirectly include this file:

Functions

- **Status compress** (char *fileName)
- FILE * **ffOpenFile** (const char *filename)

  Opens a file in rb (read binary) mode.
- size_t **flushBitWriterBuffer** (**BitWriter** *bw)

4.8.1 Function Documentation

4.8.1.1 compress()

```
 Status compress (
             char * fileName)  [extern]
```

4.8.1.2 ffOpenFile()

```
FILE * ffOpenFile (
             const char * filename)  [extern]
```

Opens a file in rb (read binary) mode.

Parameters

filename	The name of the file. (probably with absolute path)

Returns

FILE* or NULL

4.8.1.3 flushBitWriterBuffer()

```
size_t flushBitWriterBuffer (
              BitWriter * bw)  [extern]
```

4.9 compress.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/19/2025.
//

#ifndef DEFLATE_COMPRESS_H
#define DEFLATE_COMPRESS_H

#include <stddef.h>

#include "bitwriter.h"
#include "status.h"

extern Status compress(char* fileName);
extern FILE* ffOpenFile(const char* filename);
extern size_t flushBitWriterBuffer(BitWriter* bw);

#endif //DEFLATE_COMPRESS_H
```

4.10 CRC_CHECKSUM.c File Reference

`#include "CRC_CHECKSUM.h"`
`#include <stdio.h>`
Include dependency graph for CRC_CHECKSUM.c:

4.11 CRC_CHECKSUM.h File Reference

`#include <stdint.h>`
Include dependency graph for CRC_CHECKSUM.h: This graph shows which files directly or indirectly include this file:

Macros

- #define **CRC32_POLYNOMIAL** 0xEDB88320UL
- #define **CRC32_INITIAL_VALUE** 0xFFFFFFFFUL

Functions

- uint32_t **calculate_crc32** (uint32_t current_crc, const uint8_t ∗data, size_t length)

  Updates a running CRC32 checksum based on a block of data. The CRC32 algorithm used is the standard IEEE 802.3 (used in Gzip and Zlib).

4.11.1 Macro Definition Documentation

4.11.1.1 CRC32_INITIAL_VALUE

```
#define CRC32_INITIAL_VALUE 0xFFFFFFFFUL
```

4.11.1.2 CRC32_POLYNOMIAL

```
#define CRC32_POLYNOMIAL 0xEDB88320UL
```

4.11.2 Function Documentation

4.11.2.1 calculate_crc32()

```
uint32_t calculate_crc32 (
             uint32_t current_crc,
             const uint8_t * data,
             size_t length)  [extern]
```

Updates a running CRC32 checksum based on a block of data. The CRC32 algorithm used is the standard IEEE 802.3 (used in Gzip and Zlib).

Parameters

current_crc	The current running CRC value (should be 0xFFFFFFFF for the start).
data	Pointer to the buffer containing the data chunk.
length	The number of bytes in the data chunk.

Returns

uint32_t The updated CRC value.

Updates a running CRC32 checksum based on a block of data. The CRC32 algorithm used is the standard IEEE 802.3 (used in Gzip and Zlib).

- **Parameters**

current_crc	The current running CRC value (initial value 0xFFFFFFFF).
data	Pointer to the buffer containing the data chunk.
length	The number of bytes in the data chunk.

Returns

uint32_t The updated CRC value.

4.12 CRC_CHECKSUM.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/24/2025.
//

#ifndef DEFLATE_CRC_CHECKSUM_H
#define DEFLATE_CRC_CHECKSUM_H

// The standard polynomial used for Gzip/Zlib (IEEE 802.3)
#define CRC32_POLYNOMIAL 0xEDB88320UL

// The initial value for the CRC32 calculation in Gzip/Zlib is 0xFFFFFFFF
#define CRC32_INITIAL_VALUE 0xFFFFFFFFUL
#include <stdint.h>

extern uint32_t calculate_crc32(uint32_t current_crc, const uint8_t* data, size_t length);

#endif //DEFLATE_CRC_CHECKSUM_H
```

4.13 debugmalloc.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
```

Include dependency graph for debugmalloc.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **DebugmallocEntry**
- struct **DebugmallocData**

Macros

- #define **malloc**(S)
- #define **calloc**(N, S)
- #define **realloc**(P, S)
- #define **free**(P)
- #define **strdup**(S)
- #define **strndup**(S, N)

Typedefs

- typedef struct DebugmallocEntry **DebugmallocEntry**
- typedef struct DebugmallocData **DebugmallocData**

Enumerations

- enum { **debugmalloc_canary_size** = 64 , **debugmalloc_canary_char** = 'K' , **debugmalloc_tablesize** = 256 , **debugmalloc_max_block_size_default** = 1048576 }

Functions

- int **putenv** (char *)

4.13.1 Macro Definition Documentation

4.13.1.1 calloc

```
#define calloc(
              N,
              S)
```

Value:

```
debugmalloc_malloc_full((N)*(S), "calloc", #N ", " #S, __FILE__, __LINE__, true)
```

4.13.1.2 free

```
#define free(
              P)
```

Value:

```
debugmalloc_free_full((P), "free", __FILE__, __LINE__)
```

4.13.1.3 malloc

```
#define malloc(
              S)
```

Value:

```
debugmalloc_malloc_full((S), "malloc", #S, __FILE__, __LINE__, false)
```

4.13.1.4 realloc

```
#define realloc(
              P,
              S)
```

Value:

```
debugmalloc_realloc_full((P), (S), "realloc", #S, __FILE__, __LINE__)
```

4.13.1.5 strdup

```
#define strdup(
              S)
```

Value:

```
debugmalloc_strdup((S), "strdup", #S, __FILE__, __LINE__)
```

4.13.1.6 strndup

```
#define strndup(
              S,
              N)
```

Value:

```
debugmalloc_strndup((S), (N), "strndup", #S, __FILE__, __LINE__)
```

4.13.2 Typedef Documentation

4.13.2.1 DebugmallocData

```
typedef struct DebugmallocData DebugmallocData
```

4.13.2.2 DebugmallocEntry

```
typedef struct DebugmallocEntry DebugmallocEntry
```

4.13.3 Enumeration Type Documentation

4.13.3.1 anonymous enum

```
anonymous enum
```

Enumerator

debugmalloc_canary_size	
debugmalloc_canary_char	
debugmalloc_tablesize	
debugmalloc_max_block_size_default	

4.13.4 Function Documentation

4.13.4.1 putenv()

```
int putenv (
             char * )
```

4.14 debugmalloc.h

Go to the documentation of this file.

```c
#ifndef DEBUGMALLOC_H
#define DEBUGMALLOC_H

#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdarg.h>


enum {
    /* size of canary in bytes. should be multiple of largest alignment
     * required by any data type (usually 8 or 16) */
    debugmalloc_canary_size = 64,

    /* canary byte */
    debugmalloc_canary_char = 'K',

    /* hash table size for allocated entries */
    debugmalloc_tablesize = 256,

    /* max block size for allocation, can be modified with debugmalloc_max_block_size() */
    debugmalloc_max_block_size_default = 1048576
};


/* make getpid and putenv "crossplatform". deprecated on windows but they work just fine,
 * however not declared. */
#ifdef _WIN32
    /* windows */
    #include <process.h>
    #ifdef _MSC_VER
        #pragma warning(push)
        /* visual studio, getenv/getpid deprecated warning */
        #pragma warning(disable: 4996)
        #pragma warning(disable: 4127)
    #else
        /* other windows. the declaration is unfortunately hidden
         * in mingw header files by ifdefs. */
        int putenv(const char *);
    #endif
#else
    /* posix */
    #include <unistd.h>
    int putenv(char *);
#endif


/* linked list entry for allocated blocks */
typedef struct DebugmallocEntry {
    void *real_mem;     /* the address of the real allocation */
    void *user_mem;     /* address shown to the user */
    size_t size;        /* size of block requested by user */

    char file[64];      /* malloc called in this file */
    unsigned line;      /* malloc called at this line in file */
    char func[32];      /* allocation function called (malloc, calloc, realloc) */
    char expr[128];     /* expression calculating the size of allocation */

    struct DebugmallocEntry *prev, *next;  /* for doubly linked list */
} DebugmallocEntry;


/* debugmalloc singleton, storing all state */
typedef struct DebugmallocData {
    char logfile[256];    /* log file name or empty string */
    long max_block_size;  /* max size of a single block allocated */
    long alloc_count;     /* currently allocated; decreased with free */
    long long alloc_bytes;
    long all_alloc_count; /* all allocations, never decreased */
    long long all_alloc_bytes;
    DebugmallocEntry head[debugmalloc_tablesize], tail[debugmalloc_tablesize];  /* head and tail
elements of allocation lists */
} DebugmallocData;


/* this forward declaration is required by the singleton manager function */
static DebugmallocData * debugmalloc_create(void);


```

```c
/* creates singleton instance. as this function is static included to different
 * translation units, multiple instances of the static variables are created.
 * to make sure it is really a singleton, these instances must know each other
 * somethow. an environment variable is used for that purpose, ie. the address
 * of the singleton allocated is stored by the operating system.
 * this implementation is not thread-safe. */
static DebugmallocData * debugmalloc_singleton(void) {
    static char envstr[100];
    static void *instance = NULL;

    /* if we do not know the address of the singleton:
     * - maybe we are the one to create it (env variable also does not exist)
     * - or it is already created, and stored in the env variable. */
    if (instance == NULL) {
        char envvarname[100] = "";
        sprintf(envvarname, "%s%d", "debugmallocsingleton", (int) getpid());
        char *envptr = getenv(envvarname);
        if (envptr == NULL) {
            /* no env variable: create singleton. */
            instance = debugmalloc_create();
            sprintf(envstr, "%s=%p", envvarname, instance);
            putenv(envstr);
        } else {
            /* another copy of this function already created it. */
            int ok = sscanf(envptr, "%p", &instance);
            if (ok != 1) {
                fprintf(stderr, "debugmalloc: nem lehet ertelmezni: %s!\n", envptr);
                abort();
            }
        }
    }

    return (DebugmallocData *) instance;
}


/* better version of strncpy, always terminates string with \0. */
static void debugmalloc_strlcpy(char *dest, char const *src, size_t destsize) {
    strncpy(dest, src, destsize-1); /* 03.09.2025 by KZs: destsize changed to destsize-1 according to
warnings*/
    dest[destsize - 1] = '\0';
}


/* set the name of the log file for debugmalloc. empty filename
 * means logging to stderr. */
static void debugmalloc_log_file(char const *logfilename) {
    if (logfilename == NULL)
        logfilename = "";
    DebugmallocData *instance = debugmalloc_singleton();
    debugmalloc_strlcpy(instance->logfile, logfilename, sizeof(instance->logfile));
}


/* set the maximum size of one block. useful for debugging purposes. */
static void debugmalloc_max_block_size(long max_block_size) {
    DebugmallocData *instance = debugmalloc_singleton();
    instance->max_block_size = max_block_size;
}



/* printf to the log file, or stderr. */
static void debugmalloc_log(char const *format, ...) {
    DebugmallocData *instance = debugmalloc_singleton();
    FILE *f = stderr;
    if (instance->logfile[0] != '\0') {
        f = fopen(instance->logfile, "at");
        if (f == NULL) {
            f = stderr;
            fprintf(stderr, "debugmalloc: nem tudom megnyitni a %s fajlt irasra!\n",
instance->logfile);
            debugmalloc_strlcpy(instance->logfile, "", sizeof(instance->logfile));
        }
    }

    va_list ap;
    va_start(ap, format);
    vfprintf(f, format, ap);
    va_end(ap);

    if (f != stderr)
        fclose(f);
}


/* initialize a memory block allocated for the user. the start and the end
```

```
 * of the block is initialized with the canary characters. if 'zero' is
 * true, the user memory area is zero-initialized, otherwise it is also
 * filled with the canary character to simulate garbage in memory. */
static void debugmalloc_memory_init(DebugmallocEntry *elem, bool zero) {
    unsigned char *real_mem = (unsigned char *) elem->real_mem;
    unsigned char *user_mem = (unsigned char *) elem->user_mem;
    unsigned char *canary1 = real_mem;
    unsigned char *canary2 = real_mem + debugmalloc_canary_size + elem->size;
    memset(canary1, debugmalloc_canary_char, debugmalloc_canary_size);
    memset(canary2, debugmalloc_canary_char, debugmalloc_canary_size);
    memset(user_mem, zero ? 0 : debugmalloc_canary_char, elem->size);
}

/* check canary, return true if ok, false if corrupted. */
static bool debugmalloc_canary_ok(DebugmallocEntry const *elem) {
    unsigned char *real_mem = (unsigned char *) elem->real_mem;
    unsigned char *canary1 = real_mem;
    unsigned char *canary2 = real_mem + debugmalloc_canary_size + elem->size;
    for (size_t i = 0; i < debugmalloc_canary_size; ++i) {
        if (canary1[i] != debugmalloc_canary_char)
            return false;
        if (canary2[i] != debugmalloc_canary_char)
            return false;
    }
    return true;
}


/* dump memory contents to log file. */
static void debugmalloc_dump_memory(char const *mem, size_t size) {
    for (unsigned y = 0; y < (size + 15) / 16; y++) {
        char line[80];
        int pos = 0;
        pos += sprintf(line + pos, "      %04x  ", y * 16);
        for (unsigned x = 0; x < 16; x++) {
            if (y * 16 + x < size)
                pos += sprintf(line + pos, "%02x ", (unsigned char)mem[y * 16 + x]);
            else
                pos += sprintf(line + pos, "   ");
        }
        pos += sprintf(line + pos, "  ");
        for (unsigned x = 0; x < 16; x++) {
            if (y * 16 + x < size) {
                unsigned char c = mem[y * 16 + x];
                pos += sprintf(line + pos, "%c", isprint(c) ? c : '.');
            }
            else {
                pos += sprintf(line + pos, " ");
            }
        }
        debugmalloc_log("%s\n", line);
    }
}


/* dump data of allocated memory block.
 * if the canary is corrupted, it is also written to the log. */
static void debugmalloc_dump_elem(DebugmallocEntry const *elem) {
    bool canary_ok = debugmalloc_canary_ok(elem);

    debugmalloc_log("  %p, %u bajt, kanari: %s\n"
                    "  %s:%u, %s(%s)\n",
                      elem->user_mem, (unsigned) elem->size, canary_ok ? "ok" : "**SERULT**",
                      elem->file, elem->line,
                      elem->func, elem->expr);

    if (!canary_ok) {
        debugmalloc_log("    ELOTTE kanari: \n");
        debugmalloc_dump_memory((char const *) elem->real_mem, debugmalloc_canary_size);
    }

    debugmalloc_dump_memory((char const *) elem->user_mem, elem->size > 64 ? 64 : elem->size);

    if (!canary_ok) {
        debugmalloc_log("    UTANA kanari: \n");
        debugmalloc_dump_memory((char const *) elem->real_mem + debugmalloc_canary_size + elem->size, debugmalloc_canary_size);
    }
}


/* dump data of all memory blocks allocated. */
static void debugmalloc_dump(void) {
    DebugmallocData *instance = debugmalloc_singleton();
    debugmalloc_log("** DEBUGMALLOC DUMP ************************************\n");
    int cnt = 0;
    for (size_t i = 0; i < debugmalloc_tablesize; i++) {
        DebugmallocEntry *head = &instance->head[i];
        for (DebugmallocEntry *iter = head->next; iter->next != NULL; iter = iter->next) {
            ++cnt;
            debugmalloc_log("** %d/%d. rekord:\n", cnt, instance->alloc_count);
            debugmalloc_dump_elem(iter);
        }
    }
    debugmalloc_log("** DEBUGMALLOC DUMP VEGE *******************************\n");
}


/* called at program exit to dump data if there is a leak,
 * ie. allocated block remained. */
static void debugmalloc_atexit_dump(void) {
    DebugmallocData *instance = debugmalloc_singleton();

    if (instance->alloc_count > 0) {
        debugmalloc_log("\n"
                        "********************************************************\n"
                        "* MEMORIASZIVARGAS VAN A PROGRAMBAN!!!\n"
                        "********************************************************\n"
                        "\n");
        debugmalloc_dump();
    } else {
        debugmalloc_log("****************************************************\n"
                        "* Debugmalloc: nincs memoriaszivargas a programban.\n"
                        "* Osszes foglalas: %d blokk, %d bajt.\n"
                        "****************************************************\n",
                        instance->all_alloc_count, instance->all_alloc_bytes);
    }
}


/* hash function for bucket hash. */
static size_t debugmalloc_hash(void *address) {
    /* the last few bits are ignored, as they are usually zero for
     * alignment purposes. all tested architectures used 16 byte allocation. */
    size_t cut = (size_t)address » 4;
    return cut % debugmalloc_tablesize;
}


/* insert element to hash table. */
static void debugmalloc_insert(DebugmallocEntry *entry) {
    DebugmallocData *instance = debugmalloc_singleton();
    size_t idx = debugmalloc_hash(entry->user_mem);
    DebugmallocEntry *head = &instance->head[idx];
    entry->prev = head;
    entry->next = head->next;
    head->next->prev = entry;
    head->next = entry;
    instance->alloc_count += 1;
    instance->alloc_bytes += entry->size;
    instance->all_alloc_count += 1;
    instance->all_alloc_bytes += entry->size;
}


/* remove element from hash table */
static void debugmalloc_remove(DebugmallocEntry *entry) {
    DebugmallocData *instance = debugmalloc_singleton();
    entry->next->prev = entry->prev;
    entry->prev->next = entry->next;
    instance->alloc_count -= 1;
    instance->alloc_bytes -= entry->size;
}


/* find element in hash table, given with the memory address that the user sees.
 * @return the linked list entry, or null if not found. */
static DebugmallocEntry *debugmalloc_find(void *mem) {
    DebugmallocData *instance = debugmalloc_singleton();
    size_t idx = debugmalloc_hash(mem);
    DebugmallocEntry *head = &instance->head[idx];
    for (DebugmallocEntry *iter = head->next; iter->next != NULL; iter = iter->next)
        if (iter->user_mem == mem)
            return iter;
    return NULL;
}


/* allocate memory. this function is called via the macro. */
static void *debugmalloc_malloc_full(size_t size, char const *func, char const *expr, char const
*file, unsigned line, bool zero) {
    /* imitate standard malloc: return null if size is zero */
    if (size == 0)
        return NULL;
```

```c

    /* check max size */
    DebugmallocData *instance = debugmalloc_singleton();
    if (size > (size_t)(instance->max_block_size)) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: a blokk merete tul nagy, %u bajt; debugmalloc_max_block_size() fuggvennyel novelheto.\n", func, file, line, (unsigned) size);
        abort();
    }

    /* allocate more memory, make room for canary */
    void *real_mem = malloc(size + 2 * debugmalloc_canary_size);
    if (real_mem == NULL) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: nem sikerult %u meretu memoriat foglalni!\n", func, file, line, (unsigned) size);
        return NULL;
    }

    /* allocate memory for linked list element */
    DebugmallocEntry *newentry = (DebugmallocEntry *) malloc(sizeof(DebugmallocEntry));
    if (newentry == NULL) {
        free(real_mem);
        debugmalloc_log("debugmalloc: %s @ %s:%u: le tudtam foglalni %u memoriat, de utana a sajatnak nem, sry\n", func, file, line, (unsigned) size);
        abort();
    }

    /* metadata of allocation: caller function, code line etc. */
    debugmalloc_strlcpy(newentry->func, func, sizeof(newentry->func));
    debugmalloc_strlcpy(newentry->expr, expr, sizeof(newentry->expr));
    debugmalloc_strlcpy(newentry->file, file, sizeof(newentry->file));
    newentry->line = line;

    /* address of allocated memory chunk */
    newentry->real_mem = real_mem;
    newentry->user_mem = (unsigned char *) real_mem + debugmalloc_canary_size;
    newentry->size = size;
    debugmalloc_memory_init(newentry, zero);

    /* store in list and return pointer to user area */
    debugmalloc_insert(newentry);
    return newentry->user_mem;
}


/* free memory and remove list item. before deleting, the chuck is filled with
 * the canary byte to make sure that the user will see garbage if the memory
 * is accessed after freeing. */
static void debugmalloc_free_inner(DebugmallocEntry *deleted) {
    debugmalloc_remove(deleted);

    /* fill with garbage, then remove from linked list */
    memset(deleted->real_mem, debugmalloc_canary_char, deleted->size + 2 * debugmalloc_canary_size);
    free(deleted->real_mem);
    free(deleted);
}


/* free memory - called via the macro.
 * as all allocations are tracked in the list, this function can terminate the program
 * if a block is freed twice or the free function is called with an invalid address. */
static void debugmalloc_free_full(void *mem, char const *func, char const *file, unsigned line) {
    /* imitate standard free function: if ptr is null, no operation is performed */
    if (mem == NULL)
        return;

    /* find allocation, abort if not found */
    DebugmallocEntry *deleted = debugmalloc_find(mem);
    if (deleted == NULL) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: olyan teruletet probalsz felszabaditani, ami nincs lefoglalva!\n", func, file, line);
        abort();
    }

    /* check canary and then free memory */
    if (!debugmalloc_canary_ok(deleted)) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: a %p memoriateruletet tulindexelted!\n", func, file, line, mem);
        debugmalloc_dump_elem(deleted);
    }
    debugmalloc_free_inner(deleted);
}


/* realloc-like function. */
static void *debugmalloc_realloc_full(void *oldmem, size_t newsize, char const *func, char const *expr, char const *file, unsigned line) {
    /* imitate standard realloc: equivalent to free if size is null. */
    if (newsize == 0) {
        debugmalloc_free_full(oldmem, func, file, line);
        return NULL;
    }
    /* imitate standard realloc: equivalent to malloc if first param is NULL */
    if (oldmem == NULL)
        return debugmalloc_malloc_full(newsize, func, expr, file, line, 0);

    /* find old allocation. abort if not found. */
    DebugmallocEntry *oldentry = debugmalloc_find(oldmem);
    if (oldentry == NULL) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: olyan teruletet probalsz atmeretezni, ami nincs lefoglalva!\n", func, file, line);
        abort();
    }

    /* create new allocation, copy & free old data */
    void *newmem = debugmalloc_malloc_full(newsize, func, expr, file, line, false);
    if (newmem == NULL) {
        debugmalloc_log("debugmalloc: %s @ %s:%u: nem sikerult uj memoriat foglalni az atmeretezeshez!\n", func, file, line);
        /* imitate standard realloc: original block is untouched, but return NULL */
        return NULL;
    }
    size_t smaller = oldentry->size < newsize ? oldentry->size : newsize;
    memcpy(newmem, oldmem, smaller);
    debugmalloc_free_inner(oldentry);

    return newmem;
}


/* initialize debugmalloc singleton. returns the newly allocated instance */
static DebugmallocData * debugmalloc_create(void) {
    /* config check */
    if (debugmalloc_canary_size % 16 != 0) {
        debugmalloc_log("debugmalloc: a kanari merete legyen 16-tal oszthato\n");
        abort();
    }
    if (debugmalloc_canary_char == 0) {
        debugmalloc_log("debugmalloc: a kanari legyen 0-tol kulonbozo\n");
        abort();
    }
    /* avoid compiler warning if these functions are not used */
    (void) debugmalloc_realloc_full;
    (void) debugmalloc_log_file;
    (void) debugmalloc_max_block_size;

    /* create and initialize instance */
    DebugmallocData *instance = (DebugmallocData *) malloc(sizeof(DebugmallocData));
    if (instance == NULL) {
        debugmalloc_log("debugmalloc: nem sikerult elinditani a memoriakezelest\n");
        abort();
    }
    debugmalloc_strlcpy(instance->logfile, "", sizeof(instance->logfile));
    instance->max_block_size = debugmalloc_max_block_size_default;
    instance->alloc_count = 0;
    instance->alloc_bytes = 0;
    instance->all_alloc_count = 0;
    instance->all_alloc_bytes = 0;
    for (size_t i = 0; i < debugmalloc_tablesize; i++) {
        instance->head[i].prev = NULL;
        instance->head[i].next = &instance->tail[i];
        instance->tail[i].next = NULL;
        instance->tail[i].prev = &instance->head[i];
    }

    atexit(debugmalloc_atexit_dump);
    return instance;
}


/* These macro-like functions forward all allocation/free
 * calls to debugmalloc. Usage is the same, malloc(size)
 * gives the address of a new memory block, free(ptr)
 * deallocates etc.
 *
 * If you use this file, make sure that you include this
 * in *ALL* translation units (*.c) of your source. The
 * builtin free() function cannot deallocate a memory block
 * that was allocated via debugmalloc, yet the name of
 * the function is the same! */

#define malloc(S) debugmalloc_malloc_full((S), "malloc", #S, __FILE__, __LINE__, false)
#define calloc(N,S) debugmalloc_malloc_full((N)*(S), "calloc", #N ", " #S, __FILE__, __LINE__, true)
#define realloc(P,S) debugmalloc_realloc_full((P), (S), "realloc", #S, __FILE__, __LINE__)
#define free(P) debugmalloc_free_full((P), "free", __FILE__, __LINE__)
```

```

/* To include strdup and strndup functions (since c20)
 * in debugmalloc administration mechanism. */

static inline char * debugmalloc_strdup (const char *s, char const *func, char const *expr, char const
     *file, unsigned line)
{
  size_t len = strlen (s) + 1;
  void *new = debugmalloc_malloc_full (len, func, expr, file, line, 0);

  if (new == NULL)
    return NULL;

  return (char *) memcpy (new, s, len);
}


static inline char * debugmalloc_strndup (const char *s, size_t n, char const *func, char const *expr,
     char const *file, unsigned line)
{
  size_t len = strlen (s);

  if (n < len)
    len = n;

  void *new = debugmalloc_malloc_full (len+1, func, expr, file, line, 0);

  if (new == NULL)
    return NULL;

  ((char*)new)[len] = '\0';
  return (char *) memcpy (new, s, len);
}

#define strdup(S) debugmalloc_strdup((S), "strdup", #S, __FILE__, __LINE__)
#define strndup(S,N) debugmalloc_strndup((S), (N), "strndup", #S, __FILE__, __LINE__)

#if defined(_WIN32) && defined(_MSC_VER)
    #pragma warning(pop)
#endif

#endif
```

4.15 decompress.c File Reference

```
#include "status.h"
#include <stdint.h>
#include "bitreader.h"
#include "compress.h"
#include "decompress.h"
#include <stdlib.h>
#include <string.h>
#include "HUFFMAN_TABLE.h"
```

Include dependency graph for decompress.c:

Macros

- #define **ID** 0x1F8B
- #define **CM** 0x08
- #define **FLAG** 0x00
- #define **MAX_BITS** 15
- #define **CL_SYMBOLS** 19

Functions

- **Status decompress** (char *filename)

4.15.1 Macro Definition Documentation

4.15.1.1 CL_SYMBOLS

```
#define CL_SYMBOLS 19
```

4.15.1.2 CM

```
#define CM 0x08
```

4.15.1.3 FLAG

```
#define FLAG 0x00
```

4.15.1.4 ID

```
#define ID 0x1F8B
```

4.15.1.5 MAX_BITS

```
#define MAX_BITS 15
```

4.15.2 Function Documentation

4.15.2.1 decompress()

```
 Status decompress (
            char * filename)  [extern]
```

4.16 decompress.h File Reference

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

Functions

- **Status decompress** (char ∗filename)

4.16.1 Function Documentation

4.16.1.1 decompress()

```
 Status decompress (
            char * filename)  [extern]
```

4.17 decompress.h

Go to the documentation of this file.

```
//
// Created by Attila on 11/24/2025.
//

#ifndef DEFLATE_DECOMPRESS_H
#define DEFLATE_DECOMPRESS_H
extern Status decompress(char* filename);
#endif //DEFLATE_DECOMPRESS_H
```

4.18 distance.c File Reference

```
#include "distance.h"
#include <stdio.h>
```

Include dependency graph for distance.c:

Macros

- #define **NUM_DIST_CODES** 30
- #define **MAX_ALLOWED_DISTANCE** 32768

Functions

- **DISTANCE_CODE getDistanceCode** (int distance)

  Maps a raw LZ77 distance to its Deflate Symbol ID and extra bit information.

4.18.1 Macro Definition Documentation

4.18.1.1 MAX_ALLOWED_DISTANCE

```
#define MAX_ALLOWED_DISTANCE 32768
```

4.18.1.2 NUM_DIST_CODES

```
#define NUM_DIST_CODES 30
```

4.18.2 Function Documentation

4.18.2.1 getDistanceCode()

```
DISTANCE_CODE getDistanceCode (
            int distance)  [extern]
```

Maps a raw LZ77 distance to its Deflate Symbol ID and extra bit information.

- **Parameters**

distance	The raw look-back distance (1 to 32768).

Returns

DISTANCE_CODE (p. **??**) The structure containing the Symbol ID, extra bits count, and value.

4.19 distance.h File Reference

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

Data Structures

- struct **DISTANCE_CODE**

Functions

- **DISTANCE_CODE getDistanceCode** (int distance)
  Maps a raw LZ77 distance to its Deflate Symbol ID and extra bit information.

4.19.1 Function Documentation

4.19.1.1 getDistanceCode()

```
DISTANCE_CODE getDistanceCode (
            int distance)  [extern]
```

Maps a raw LZ77 distance to its Deflate Symbol ID and extra bit information.

- **Parameters**

distance	The raw look-back distance (1 to 32768).

Returns

DISTANCE_CODE (p. **??**) The structure containing the Symbol ID, extra bits count, and value.

4.20 distance.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/19/2025.
//

#ifndef DEFLATE_DISTANCE_H
#define DEFLATE_DISTANCE_H

typedef struct {
    unsigned short usSymbolID;
    int iExtraBits;
    int iExtraValue;
} DISTANCE_CODE;

extern DISTANCE_CODE getDistanceCode(int distance);

#endif //DEFLATE_DISTANCE_H
```

4.21 HUFFMAN_TABLE.c File Reference

```
#include "HUFFMAN_TABLE.h"
#include "bitreader.h"
#include "node.h"
```
Include dependency graph for HUFFMAN_TABLE.c:

Macros

- #define **INVALID_NODE_SYMBOL** 286
- #define **MAX_BITS** 15

Functions

- uint16_t **decode_symbol** (**BIT_READER** ∗reader, const **HuffmanTree** ∗tree)
- void **buildFastLookupTable** (const **HUFFMAN_CODE** ∗canonical_codes, int total_symbols, **Huffman←Entry** ∗lookup_table)
- void **buildCodeLookupTable** (**Node** ∗node, **HUFFMAN_CODE** ∗table, uint16_t current_code, int depth)

4.21.1 Macro Definition Documentation

4.21.1.1 INVALID_NODE_SYMBOL

```
#define INVALID_NODE_SYMBOL 286
```

4.21.1.2 MAX_BITS

```
#define MAX_BITS 15
```

4.21.2 Function Documentation

4.21.2.1 buildCodeLookupTable()

```
void buildCodeLookupTable (
             Node * node,
             HUFFMAN_CODE * table,
             uint16_t current_code,
             int depth)  [extern]
```

4.21.2.2 buildFastLookupTable()

```
void buildFastLookupTable (
             const  HUFFMAN_CODE * canonical_codes,
             int total_symbols,
             HuffmanEntry * lookup_table)  [extern]
```

4.21.2.3 decode_symbol()

```
uint16_t decode_symbol (
             BIT_READER * reader,
             const  HuffmanTree * tree)
```

4.22 HUFFMAN_TABLE.h File Reference

```
#include <stdint.h>
#include "bitreader.h"
#include "node.h"
```

Include dependency graph for HUFFMAN_TABLE.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **HUFFMAN_CODE**
- struct **HuffmanEntry**
- struct **CanonicalCode**
- struct **HuffmanTree**

Macros

- #define **FAST_BITS** 9
- #define **FAST_SIZE** (1 << **FAST_BITS**)
- #define **MAX_CODE_SYMBOLS** 286

Functions

- void **buildCodeLookupTable** (**Node** ∗node, **HUFFMAN_CODE** ∗table, uint16_t current_code, int depth)
- void **buildFastLookupTable** (const **HUFFMAN_CODE** ∗canonical_codes, int total_symbols, **Huffman←Entry** ∗lookup_table)
- uint16_t **decode_symbol** (**BIT_READER** ∗reader, const **HuffmanTree** ∗tree)

4.22.1 Macro Definition Documentation

4.22.1.1 FAST_BITS

```
#define FAST_BITS 9
```

4.22.1.2 FAST_SIZE

```
#define FAST_SIZE (1 <<  FAST_BITS)
```

4.22.1.3 MAX_CODE_SYMBOLS

```
#define MAX_CODE_SYMBOLS 286
```

4.22.2 Function Documentation

4.22.2.1 buildCodeLookupTable()

```
void buildCodeLookupTable (
              Node * node,
              HUFFMAN_CODE * table,
             uint16_t current_code,
             int depth)  [extern]
```

4.22.2.2 buildFastLookupTable()

```
void buildFastLookupTable (
             const  HUFFMAN_CODE * canonical_codes,
             int total_symbols,
              HuffmanEntry * lookup_table)  [extern]
```

4.22.2.3 decode_symbol()

```
uint16_t decode_symbol (
              BIT_READER * reader,
             const  HuffmanTree * tree)
```

4.23 HUFFMAN_TABLE.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/23/2025.
//

#ifndef DEFLATE_HUFFMAN_TABLE_H
#define DEFLATE_HUFFMAN_TABLE_H

#include <stdint.h>

#include "bitreader.h"
#include "node.h"

typedef struct {
    uint16_t code; //for example 101 in binary which means right, left, right in the tree
    uint8_t length; //in this example 3 which says how many bits are there
} HUFFMAN_CODE;

#define FAST_BITS 9
#define FAST_SIZE (1 « FAST_BITS) // 512 entries
#define MAX_CODE_SYMBOLS 286 // Max symbols for T_LL (the largest tree)

typedef struct {
    uint16_t symbol; // The decoded symbol
    uint8_t bits;   // Number of bits consumed (Length)
} HuffmanEntry;

// --- 2. Full Code/Length Storage (for the Slow Path) ---
// This stores the mathematically generated canonical codes for *all* symbols.
```

```
typedef struct {
    uint16_t code;   // The Canonical Code value (must be bit-reversed if using LSB-first reading)
    uint8_t length; // The length of the code (up to 15)
} CanonicalCode;

// --- 3. The Unified Tree Structure ---
typedef struct {
    // I. Fast Lookup Table: Resolves all codes <= FAST_BITS
    HuffmanEntry lookup_table[1 « FAST_BITS];

    // II. Storage for Long Codes (The Slow Path Data):
    // Used to resolve codes longer than FAST_BITS.
    // This allows the slow path in decode_symbol to check all possible codes.
    CanonicalCode codes_list[MAX_CODE_SYMBOLS];

    // III. Metadata
    uint16_t total_symbols; // e.g., 19, HLIT, or HDIST+1
    uint8_t max_length;     // Max code length observed in this tree (up to 15)

} HuffmanTree;


extern void buildCodeLookupTable(Node* node, HUFFMAN_CODE* table, uint16_t current_code, int depth);

extern void buildFastLookupTable(
    const HUFFMAN_CODE* canonical_codes,
    int total_symbols,
    HuffmanEntry* lookup_table
);
uint16_t decode_symbol(BIT_READER* reader, const HuffmanTree* tree);
#endif //DEFLATE_HUFFMAN_TABLE_H
```

4.24 length.c File Reference

```
#include "length.h"
#include <stdio.h>
```

Include dependency graph for length.c:

Macros

- #define **MIN_MATCH_LENGTH** 3
- #define **MAX_MATCH_LENGTH** 258
- #define **LITERAL_LENGTH_CODE_START** 257
- #define **NUM_LENGTH_CODES** 29

Functions

- **LENGTH_CODE getLengthCode** (int length)

  LengthCode.

4.24.1 Macro Definition Documentation

4.24.1.1 LITERAL_LENGTH_CODE_START

```
#define LITERAL_LENGTH_CODE_START 257
```

4.24.1.2 MAX_MATCH_LENGTH

```
#define MAX_MATCH_LENGTH 258
```

4.24.1.3 MIN_MATCH_LENGTH

```
#define MIN_MATCH_LENGTH 3
```

4.24.1.4 NUM_LENGTH_CODES

```
#define NUM_LENGTH_CODES 29
```

4.24.2 Function Documentation

4.24.2.1 getLengthCode()

```
 LENGTH_CODE getLengthCode (
             int length)  [extern]
```

LengthCode.

The getLengthCode function calculates the Length Code for each match length, and returns a struct with the necessary data in it. For more information on Length Code's check out the official rfc1951 standard documentation. (`https://datatracker.ietf.org/doc/html/rfc1951#page-11`)

Parameters

length	The actual length between (3 and 258)

Returns

LengthCode struct which stores the length code, the required extra bits, and then the extra value in those extra bits,

4.25 length.h File Reference

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

Data Structures

- struct **LENGTH_CODE**

Functions

- **LENGTH_CODE getLengthCode** (int length)
  LengthCode.

4.25.1 Function Documentation

4.25.1.1 getLengthCode()

```
LENGTH_CODE getLengthCode (
            int length)  [extern]
```

LengthCode.

The getLengthCode function calculates the Length Code for each match length, and returns a struct with the necessary data in it. For more information on Length Code's check out the official rfc1951 standard documentation. (`https://datatracker.ietf.org/doc/html/rfc1951#page-11`)

Parameters

length	The actual length between (3 and 258)

Returns

LengthCode struct which stores the length code, the required extra bits, and then the extra value in those extra bits,

4.26 length.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/19/2025.
//

#ifndef DEFLATE_LENGTH_H
#define DEFLATE_LENGTH_H

typedef struct {
    unsigned short usSymbolID;
    int iExtraBits;
    int iExtraValue;
} LENGTH_CODE;


extern LENGTH_CODE getLengthCode(int length);

#endif //DEFLATE_LENGTH_H
```

4.27 LZ77.c File Reference

```
#include <stdint.h>
#include "LZ77.h"
```

Include dependency graph for LZ77.c:

Functions

- **LZ77_compressed createLiteralLZ77** (const uint8_t byte)

  Create Literal Struct for LZ77 compression.
- **LZ77_compressed createMatchLZ77** (const uint16_t distance, const uint16_t length)

  Create MATCH Struct for LZ77 compression.
- **LZ77_buffer** ∗ **initLZ77Buffer** (void)

  *Initalizes the **LZ77_buffer** (p. **??**) struct.*
- void **expandBuffer** (**LZ77_buffer** ∗buffer)

  Expands the buffer capacity by EXPAND_BY tokens.
- void **appendToken** (**LZ77_buffer** ∗buffer, const **LZ77_compressed** token)

  Appends a token to the buffer.
- void **freeLZ77Buffer** (**LZ77_buffer** ∗buffer)

  Frees all dynamically allocated memory associated with the buffer.

4.27.1 Function Documentation

4.27.1.1 appendToken()

```
void appendToken (
              LZ77_buffer * buffer,
             const  LZ77_compressed token)  [extern]
```

Appends a token to the buffer.

Parameters

buffer	The buffer to append the token to.

Returns

void

4.27.1.2 createLiteralLZ77()

```
LZ77_compressed createLiteralLZ77 (
            const uint8_t byte) [extern]
```

Create Literal Struct for LZ77 compression.

The createLiteralLZ77 function allocates memory for one single LZ77_compresed struct filled with enum LITERAL and the actual byte.

Parameters

byte	The actual byte to be written into the file

Returns

LZ77_compressed (p. **??**)

4.27.1.3 createMatchLZ77()

```
LZ77_compressed createMatchLZ77 (
            const uint16_t distance,
            const uint16_t length) [extern]
```

Create MATCH Struct for LZ77 compression.

The createMatchLZ77 function allocates memory for one single LZ77_compresed struct filled with enum MATCH and the actual distance / length pair.

Parameters

distance	The distance from the last occurrence
length	The length which specifies the length from last occurrence

Returns

LZ77_compressed (p. **??**)

4.27.1.4 expandBuffer()

```
void expandBuffer (
             LZ77_buffer * buffer) [extern]
```

Expands the buffer capacity by EXPAND_BY tokens.

Parameters

buffer	The buffer to expand.

Returns

void

4.27.1.5 freeLZ77Buffer()

```
void freeLZ77Buffer (
              LZ77_buffer * buffer)  [extern]
```

Frees all dynamically allocated memory associated with the buffer.

Parameters

buffer	The buffer structure to be freed.

Returns

void

4.27.1.6 initLZ77Buffer()

```
 LZ77_buffer * initLZ77Buffer (
             void )  [extern]
```

Initalizes the **LZ77_buffer** (p. **??**) struct.

Allocates initial memory for the token array.

Returns

LZ77_buffer∗ The location in memory. MUST BE FREED afterward!

4.28 LZ77.h File Reference

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

Include dependency graph for LZ77.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **LZ77_compressed**

  The fundamental LZ77 token structure.
- struct **LZ77_buffer**

  Structure to manage a dynamic array (growing buffer) of LZ77 tokens.

Macros

- #define **EXPAND_BY** 50

  The amount to expand the buffer's capacity when full.

Enumerations

- enum **LZ77_encoded_type** { **LITERAL** , **MATCH** }

  Enumeration to distinguish between a literal byte and a match pair.

Functions

- **LZ77_compressed createLiteralLZ77** (const uint8_t byte)

  Create Literal Struct for LZ77 compression.
- **LZ77_compressed createMatchLZ77** (const uint16_t distance, const uint16_t length)

  Create MATCH Struct for LZ77 compression.
- **LZ77_buffer** ∗ **initLZ77Buffer** (void)

  *Initalizes the **LZ77_buffer** (p. **??**) struct.*
- void **expandBuffer** (**LZ77_buffer** ∗buffer)

  Expands the buffer capacity by EXPAND_BY tokens.
- void **appendToken** (**LZ77_buffer** ∗buffer, **LZ77_compressed** token)

  Appends a token to the buffer.
- void **freeLZ77Buffer** (**LZ77_buffer** ∗buffer)

  Frees all dynamically allocated memory associated with the buffer.

4.28.1 Macro Definition Documentation

4.28.1.1 EXPAND_BY

```
#define EXPAND_BY 50
```

The amount to expand the buffer's capacity when full.

4.28.2 Enumeration Type Documentation

4.28.2.1 LZ77_encoded_type

```
enum LZ77_encoded_type
```

Enumeration to distinguish between a literal byte and a match pair.

Enumerator

LITERAL	
MATCH	

4.28.3 Function Documentation

4.28.3.1 appendToken()

```
void appendToken (
              LZ77_buffer * buffer,
             const  LZ77_compressed token)
```

Appends a token to the buffer.

Parameters

buffer	The buffer to append the token to.
token	The **LZ77_compressed** (p. **??**) token.
buffer	The buffer to append the token to.

Returns

void

4.28.3.2 createLiteralLZ77()

```
LZ77_compressed createLiteralLZ77 (
            const uint8_t byte)
```

Create Literal Struct for LZ77 compression.

The createLiteralLZ77 function allocates memory for one single LZ77_compresed struct filled with enum LITERAL and the actual byte.

Parameters

byte	The actual byte to be written into the file

Returns

LZ77_compressed∗ The location in memory. MUST BE FREED afterward!

The createLiteralLZ77 function allocates memory for one single LZ77_compresed struct filled with enum LITERAL and the actual byte.

Parameters

byte	The actual byte to be written into the file

Returns

LZ77_compressed (p. **??**)

4.28.3.3 createMatchLZ77()

```
LZ77_compressed createMatchLZ77 (
            const uint16_t distance,
            const uint16_t length)
```

Create MATCH Struct for LZ77 compression.

The createMatchLZ77 function allocates memory for one single LZ77_compresed struct filled with enum MATCH and the actual distance / length pair.

Parameters

distance	The distance from the last occurrence
length	The length which specifies the length from last occurrence

Returns

LZ77_compressed∗ The location in memory. MUST BE FREED afterward!

The createMatchLZ77 function allocates memory for one single LZ77_compresed struct filled with enum MATCH and the actual distance / length pair.

Parameters

distance	The distance from the last occurrence
length	The length which specifies the length from last occurrence

Returns

LZ77_compressed (p. **??**)

4.28.3.4 expandBuffer()

```
void expandBuffer (
              LZ77_buffer * buffer)
```

Expands the buffer capacity by EXPAND_BY tokens.

Parameters

buffer	The buffer to expand.
buffer	The buffer to expand.

Returns

void

4.28.3.5 freeLZ77Buffer()

```
void freeLZ77Buffer (
              LZ77_buffer * buffer)
```

Frees all dynamically allocated memory associated with the buffer.

Parameters

buffer	The buffer structure to be freed.
buffer	The buffer structure to be freed.

Returns

void

4.28.3.6 initLZ77Buffer()

```
LZ77_buffer * initLZ77Buffer (
            void )
```

Initalizes the **LZ77_buffer** (p. **??**) struct.

Allocates initial memory for the token array.

Returns

LZ77_buffer∗ The location in memory. MUST BE FREED afterward!

4.29 LZ77.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/16/2025.
//
#ifndef DEFLATE_LZ77_H
#define DEFLATE_LZ77_H

#include <stdint.h>
#include <stdlib.h>

#define EXPAND_BY 50

typedef enum {
    LITERAL,
    MATCH
} LZ77_encoded_type;

typedef struct {
    LZ77_encoded_type type;

    union {
        uint8_t literal;
        struct {
            uint16_t distance;
            uint16_t length;
        } match;
    } data;
} LZ77_compressed;

typedef struct {
    LZ77_compressed* tokens;
    size_t size;
    size_t capacity;
} LZ77_buffer;

LZ77_compressed createLiteralLZ77(const uint8_t byte);

LZ77_compressed createMatchLZ77(const uint16_t distance, const uint16_t length);


// --- Buffer Management Functions ---

LZ77_buffer* initLZ77Buffer(void);

void expandBuffer(LZ77_buffer* buffer);

void appendToken(LZ77_buffer* buffer, LZ77_compressed token);

void freeLZ77Buffer(LZ77_buffer* buffer);

#endif //DEFLATE_LZ77_H
```

4.30 main.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "compress.h"
#include "decompress.h"
#include "LZ77.h"
#include "distance.h"
#include "length.h"
#include "status.h"
#include "bitwriter.h"
```

Include dependency graph for main.c:

Functions

- int **main** (int argc, char **argv)

4.30.1 Function Documentation

4.30.1.1 main()

```
int main (
            int argc,
            char ** argv)
```

4.31 node.c File Reference

```
#include "node.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "debugmalloc.h"
```

Include dependency graph for node.c:

Macros

- #define **INVALID_NODE_SYMBOL** 286

Functions

- **Node** ∗ **createNode** (unsigned short usSymbol, int freq)
- void **freeTree** (**Node** ∗top)
- void **compressCodeLengths** (const uint8_t ∗all_lengths, size_t count, uint8_t ∗compressed_lengths, uint16_t ∗cl_frequencies, uint8_t ∗extra_bits_values, size_t ∗compressed_count)
- void **findCodeLengthsInTree** (**Node** ∗node, uint8_t ∗lengths, uint8_t depth)
- **MinHeap** ∗ **createMinHeap** (int capacity)
- void **addToMinHeap** (**MinHeap** ∗minHeap, **Node** ∗node)
- void **swapNodePointers** (**Node** ∗∗a, **Node** ∗∗b)
- void **minHeapify** (**MinHeap** ∗minHeap, int i)
- **Node** ∗ **extractMin** (**MinHeap** ∗minHeap)
- void **freeMinHeap** (**MinHeap** ∗minHeap)
- void **printHeap** (**MinHeap** ∗minHeap)
- void **buildMinHeap** (**MinHeap** ∗minHeap)
- int **parentIndex** (int i)
- int **leftChildIndex** (int i)
- void **insertMinHeap** (**MinHeap** ∗minHeap, **Node** ∗newNode)
- **Node** ∗ **buildHuffmanTree** (**MinHeap** ∗minHeap)

  Creates the Huffman tree from the populated Min-Heap.
- void **extract_code_lengths** (**Node** ∗npCurrent, uint8_t uiCurrentDepth, uint8_t ∗uiLengthCodes)

  Traverses a Huffman tree to determine the bit length (depth) for every symbol.

4.31.1 Macro Definition Documentation

4.31.1.1 INVALID_NODE_SYMBOL

```
#define INVALID_NODE_SYMBOL 286
```

4.31.2 Function Documentation

4.31.2.1 addToMinHeap()

```
void addToMinHeap (
              MinHeap * minHeap,
              Node * node)  [extern]
```

4.31.2.2 buildHuffmanTree()

```
 Node * buildHuffmanTree (
              MinHeap * minHeap)
```

Creates the Huffman tree from the populated Min-Heap.

- This is the greedy algorithm core: repeatedly combine the two smallest nodes.

- **Parameters**

minHeap	The initialized Min-Heap containing all leaf nodes.

Returns

Node∗ The root of the completed Huffman tree.

4.31.2.3 buildMinHeap()

```
void buildMinHeap (
              MinHeap * minHeap)
```

4.31.2.4 compressCodeLengths()

```
void compressCodeLengths (
             const uint8_t * all_lengths,
             size_t count,
             uint8_t * compressed_lengths,
             uint16_t * cl_frequencies,
             uint8_t * extra_bits_values,
             size_t * compressed_count)  [extern]
```

4.31.2.5 createMinHeap()

```
 MinHeap * createMinHeap (
             int capacity)  [extern]
```

4.31.2.6 createNode()

```
 Node * createNode (
             unsigned short usSymbol,
             int freq)
```

4.31.2.7 extract_code_lengths()

```
void extract_code_lengths (
              Node * npCurrent,
             uint8_t uiCurrentDepth,
             uint8_t * uiLengthCodes)  [extern]
```

Traverses a Huffman tree to determine the bit length (depth) for every symbol.

-

Parameters

current_node	The current node in the traversal (start with the tree root).
uiCurrentDepth	The depth of the current node (start with 0 for the root).
uiLengthCodes	The array where the resulting code lengths are stored.

4.31.2.8 extractMin()

```
Node * extractMin (
            MinHeap * minHeap)  [extern]
```

4.31.2.9 findCodeLengthsInTree()

```
void findCodeLengthsInTree (
            Node * node,
            uint8_t * lengths,
            uint8_t depth)  [extern]
```

4.31.2.10 freeMinHeap()

```
void freeMinHeap (
            MinHeap * minHeap)  [extern]
```

4.31.2.11 freeTree()

```
void freeTree (
            Node * top)  [extern]
```

4.31.2.12 insertMinHeap()

```
void insertMinHeap (
            MinHeap * minHeap,
            Node * newNode)
```

4.31.2.13 leftChildIndex()

```
int leftChildIndex (
            int i)
```

4.31.2.14 minHeapify()

```
void minHeapify (
            MinHeap * minHeap,
            int i)
```

4.31.2.15 parentIndex()

```
int parentIndex (
             int i)
```

4.31.2.16 printHeap()

```
void printHeap (
              MinHeap * minHeap)
```

4.31.2.17 swapNodePointers()

```
void swapNodePointers (
              Node ** a,
              Node ** b)
```

4.32 node.h File Reference

```
#include <stdint.h>
```

Include dependency graph for node.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct **Node**
- struct **MinHeap**

Typedefs

- typedef struct Node **Node**

Functions

- void **compressCodeLengths** (const uint8_t *all_lengths, size_t count, uint8_t *compressed_lengths, uint16_t *cl_frequencies, uint8_t *extra_bits_values, size_t *compressed_count)
- void **findCodeLengthsInTree** (**Node** *node, uint8_t *lengths, uint8_t depth)
- **MinHeap** * **createMinHeap** (int capacity)
- void **addToMinHeap** (**MinHeap** *minHeap, **Node** *node)
- void **printHeap** (**MinHeap** *minHeap)
- **Node** * **extractMin** (**MinHeap** *minHeap)
- **Node** * **buildHuffmanTree** (**MinHeap** *minHeap)

  Creates the Huffman tree from the populated Min-Heap.
- void **buildMinHeap** (**MinHeap** *minHeap)
- void **freeMinHeap** (**MinHeap** *minHeap)
- **Node** * **createNode** (unsigned short usSymbol, int freq)
- void **freeTree** (**Node** *top)

4.32.1 Typedef Documentation

4.32.1.1 Node

```
typedef struct Node Node
```

4.32.2 Function Documentation

4.32.2.1 addToMinHeap()

```
void addToMinHeap (
              MinHeap * minHeap,
              Node * node)  [extern]
```

4.32.2.2 buildHuffmanTree()

```
 Node * buildHuffmanTree (
              MinHeap * minHeap)  [extern]
```

Creates the Huffman tree from the populated Min-Heap.

- This is the greedy algorithm core: repeatedly combine the two smallest nodes.
-

Parameters

minHeap	The initialized Min-Heap containing all leaf nodes.

Returns

Node∗ The root of the completed Huffman tree.

4.32.2.3 buildMinHeap()

```
void buildMinHeap (
              MinHeap * minHeap)  [extern]
```

4.32.2.4 compressCodeLengths()

```
void compressCodeLengths (
             const uint8_t * all_lengths,
             size_t count,
             uint8_t * compressed_lengths,
             uint16_t * cl_frequencies,
             uint8_t * extra_bits_values,
             size_t * compressed_count)  [extern]
```

4.32.2.5 createMinHeap()

```
MinHeap * createMinHeap (
            int capacity)  [extern]
```

4.32.2.6 createNode()

```
Node * createNode (
            unsigned short usSymbol,
            int freq)  [extern]
```

4.32.2.7 extractMin()

```
Node * extractMin (
             MinHeap * minHeap)  [extern]
```

4.32.2.8 findCodeLengthsInTree()

```
void findCodeLengthsInTree (
             Node * node,
            uint8_t * lengths,
            uint8_t depth)  [extern]
```

4.32.2.9 freeMinHeap()

```
void freeMinHeap (
             MinHeap * minHeap)  [extern]
```

4.32.2.10 freeTree()

```
void freeTree (
             Node * top)  [extern]
```

4.32.2.11 printHeap()

```
void printHeap (
             MinHeap * minHeap)  [extern]
```

4.33 node.h

Go to the documentation of this file.

```
//
// Created by Attila Arnóczki on 10/18/2025.
// Node version 0.0.1
//

#ifndef HUFFMAN_NODE_H
#define HUFFMAN_NODE_H

#include <stdint.h>

typedef struct Node {
    int iFrequency;
    unsigned short usSymbol;
    struct Node* pnLeft;
    struct Node* pnRight;
} Node;

typedef struct {
    int iSize;
    int iCapacity;
    Node** ppnArray;
} MinHeap;

extern void compressCodeLengths(
    const uint8_t* all_lengths,
    size_t count,
    uint8_t* compressed_lengths, // Output buffer for RLE symbols (0-18)
    uint16_t* cl_frequencies,    // Output array of size 19
    uint8_t* extra_bits_values,  // Output buffer for RLE extra bit values
    size_t* compressed_count     // Final count of symbols generated
);

extern void findCodeLengthsInTree(Node* node, uint8_t* lengths, uint8_t depth);

extern MinHeap* createMinHeap(int capacity);

extern void addToMinHeap(MinHeap* minHeap, Node* node);

extern void printHeap(MinHeap* minHeap);

extern Node* extractMin(MinHeap* minHeap);

extern Node* buildHuffmanTree(MinHeap* minHeap);

extern void buildMinHeap(MinHeap* minHeap);

extern void freeMinHeap(MinHeap* minHeap);

extern Node* createNode(unsigned short usSymbol, int freq);

extern void freeTree(Node* top);

#endif //HUFFMAN_NODE_H
```

4.34 status.h File Reference

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

Data Structures

- struct **Status**

Enumerations

- enum **StatusCode** {
  COMPRESSION_SUCCESS , **COMPRESSION_FAILED** , **CANT_OPEN_FILE** , **CANT_ALLOCATE_↩ MEMORY** ,
  DECOMPRESS_SUCCESS , **DECOMPRESS_FAILED** }

4.34.1 Enumeration Type Documentation

4.34.1.1 StatusCode

`enum StatusCode`

Enumerator

COMPRESSION_SUCCESS	
COMPRESSION_FAILED	
CANT_OPEN_FILE	
CANT_ALLOCATE_MEMORY	
DECOMPRESS_SUCCESS	
DECOMPRESS_FAILED	

4.35 status.h

Go to the documentation of this file.

```
//
// Created by Rendszergazda on 11/19/2025.
//

#ifndef DEFLATE_STATUS_H
#define DEFLATE_STATUS_H

typedef enum {
    COMPRESSION_SUCCESS,
    COMPRESSION_FAILED,
    CANT_OPEN_FILE,
    CANT_ALLOCATE_MEMORY,
    DECOMPRESS_SUCCESS,
    DECOMPRESS_FAILED,
} StatusCode;

typedef struct {
    StatusCode code;
    char* message;
} Status;

#endif //DEFLATE_STATUS_H
```