DEV’S MANUAL

HOW DOES IT WORK?

This is a command-line tool which compresses files using Huffman algorithm.

1. Logic

Here are some brief explanations of the logic of program’s basic operations:

1. **Compressing a file.**
   - Program opens and reads the file in files/input folder, puts very unique character into an array and increase their frequency if they are already in the array.
   - Using Huffman algorithm, program builds a tree using these frequencies.
   - A **header data** containing file’s name, name size, original length, last character and the frequency array will be written in the beginning of the file.
   - Program will create/open the new compressed file in binary mode and starts to write 8-bit Huffman codes which has generated using created tree in the previous step, according to the original content.

You can add this line to the code to enable debug comments: ***#define DEBUG***

2. **Extracting a file.**
   - Program opens the file in */files/input* folder.
   - Header will be read first and it will contain the file name which is the same as before compressing.
   - According to the frequency array in header, program will build a tree.
   - New Huffman codes will be generated after building the tree.
   - Context will be converted into characters in header according to the Huffman codes.

About Huffman algorithm and coding:

Wikipedia - Huffman coding

Huffman Tree Generator

2. Header Files (includes)

1. w_.h contains useful functions.

```
void wait(); // waits for user to press Enter, then continues
void wait_after_scanf();// fix for scanf, waits for user to press Enter, then continues
void printlines(int count); // prints count times empty lines on screen
char *add_char_arrays(char *arr1, size_t size1, char *arr2, size_t size2); // adds arr1 and arr2 arrays.
char *add_char_to_array(char *arr1, size_t size, char a); // adds character to end of the character array
char *change_char_array_size(char *arr, size_t size, size_t newsize); // changes allocated memory space for char
array arr from size to newsize.
int valid_fname(char *filename); // checks if the input is a valid file name (wasn’t implemented in the current version)
int terminate(char *err); // shows the error message and returns 1 (error)
int power(int c,int e); // calculates c^e
```

2. **w_def.h** contains definitions and structure models.

3. **w_freq.h** contains frequency structure and functions.

   Frequency type structure contains 2 values.
   - char
   - frequency

   Frequency *sort_freq_array(Frequency *arr, size_t size)
   // sorts given frequency array *and returns the new one*

   Frequency *change_freq_array_size(Frequency *arr, size_t size, size_t newsize)
   // changes frequency array's size

   Frequency *fix_array(Frequency *arr, size_t size)
   // due to a problem about the last character in frequency array, while creating the header, program uses this fix in order to successfully read and extract the data without data loss. It simplty adds one more frequency with 0 values

   Frequency *read_fix_array(Frequency *arr, size_t size)
   // when extracting the file, fix entry is being removed from the frequency array in header

   int containsChar(Frequency *arr, char ch, size_t array_size)
   // returns 1 if *arr is containing ch* and 0 if it's not.

4. **w_huffman.h** contains Huffman compression functions.

   char *get_code(int a, int buff_size)
   // returns a 8 character long array containing *buff_size* bit long number *a (which is a generated Huffman code)*
   int index_of_character(struct HuffmanCodes **array, size_t size, char chr)
   // returns the index of character from HuffmanCodes array
   struct Node *create_node(char chr, int freq, struct Node *left, struct Node *right)
   // allocates memory for a new node and returns the pointer
   struct Tree *freq_to_heap(Frequency *arr,size_t size)
   // creates a Tree structure and puts the frequency array data into the tree's array
   struct Node *get_min_node(struct Tree* tree)
   // returns the node with least frequency in the tree's array and sets it's values to 0 (it will be deleted later)
   struct Tree *add_to_tree(struct Tree* tree, struct Node *newnode)
   // adds a node to the correct place in tree and returns the new tree
   struct Tree* create_tree(size_t size, struct Tree* prev, Frequency **arr, char firstchar, int freq)
   // finds 2 nodes in tree array which has the least frequencies and creates a new node containing these 2 nodes as left and right. Repeats the process n-1 times and returns the built tree (head)
   struct HuffmanCodes **generate_huffman_codes(struct Node *head,size_t size, Frequency **arr)
   // generates_huffman_codes by traversing the tree
   void *traverse_tree(struct HuffmanCodes **array, struct Node *node,int buff, int buff_size)
   // traverses tree and adds found results to HuffmanCodes array. This array contains all of the characters with their codes after this process is done
   void free_tree(struct Node *node)
   // traverses the tree and frees every node

void read_header(FILE *f, char **filename, Frequency **array, size_t *array_size, int *file_length, char *last_char)

// reads the header data when extracting the file

void create_header(FILE *f, char filename[], size_t fname_size, Frequency **array, size_t array_size, int file_length, char last_char)

// creates the header data when compressing the file

int compress(char *context, size_t length, char filename[], size_t name_size, char *filepath, struct HuffmanCodes **array, Frequency **freq_array, size_t size)

// starts the compressing operation

int decompress(char *filename, size_t *length, char **content, char **new_file)

// starts the decompressing operation and writes the result to ******content* array. This string will be written in **new_file* file.

3. Main File (huffman.c)

.There Is no forward declaration in the code. You can find main() function at the end of the file. Program is based on constantly showing the main menu and waiting for user input.

int read_file(char *filename, size_t *length, char **content, Frequency** freq_array)

// reads the file to be compressed, writes it's content to the **content* character array, builds a frequency array and puts into the **freq_array*

int load_lang()

// loads the language file and language strings array *(located in /lang)*

int compress_file(char filename[], int name_size)

// this function is called when user selects compression option and types a file name. function stores the time from beginning of the operation and prints the spent time in he end.

int extract_file(char *filename, int name_size)

// this function is called when user selects extraction option and types a file name. function stores the time from beginning of the operation and prints the spent time in the end.

int show_about()

// shows development information

int show_main_menu()

// shows main menu with 4 options