Aditya Agrawal
aagraw14@ucsc.edu

CSE13s: Spring 2021
Assignment 6: Huffman Coding
Design Document

This program will compress files by implementing an encoder that will find and return the Huffman encoding of the file that is input and also decompress files returning them to their original uncompressed size. This will be accomplished through the use of many ADTs including nodes, priority queue, codes, and stacks.

encode.c:
This file will implement the encoder with a main function and supported command line options

decode.c:
This file will implement the decoder with a main function and supported command line options

node.c:
This file will implement the node ADT that will comprise the nodes of the Huffman tree

```
typedef struct Node Node;

struct Node {
    Node *left;          // Pointer to left child.
    Node *right;         // Pointer to right child.
    uint8_t symbol;      // Node's symbol.
    uint64_t frequency; // Frequency of symbol.
};
```

Node *node_create(uint8_t symbol, uint64_t frequency);

void node_delete(Node **n);

Node *node_join(Node *left, Node *right);

void node_print(Node *n);

pq.c:
This file will implement a priority queue for the nodes which will be used by the encoder.
PriorityQueue *pq_create(uint32_t capacity);

void pq_delete(PriorityQueue **q);

bool pq_empty(PriorityQueue *q);

bool pq_full(PriorityQueue *q);

uint32_t pq_size(PriorityQueue *q);

bool enqueue(PriorityQueue *q, Node *n);

bool dequeue(PriorityQueue *q, Node **n);

void pq_print(PriorityQueue *q);

code.c:
This file will implement an ADT that will be used to maintain a stack of bits while traversing the tree in order to create a code for each symbol.

```
typedef struct Code {
    uint32_t top;
    uint8_t bits[MAX_CODE_SIZE];
} Code;
```

Code code_init(void);

uint32_t code_size(Code *c);

bool code_empty(Code *c);

bool code_full(Code *c);

bool code_push_bit(Code *c, uint8_t bit);

bool code_pop_bit(Code *c, uint8_t *bit);

void code_print(Code *c);

io.c:

This file will contain the implementation of the input/output like opening, closing, reading, and writing files and will be used by both the encoder and decoder
extern uint64_t bytes_read;
extern uint64_t bytes_written;

int read_bytes(int infile, uint8_t *buf, int nbytes);

int write_bytes(int outfile, uint8_t *buf, int nbytes);

bool read_bit(int infile, uint8_t *bit);

void write_code(int outfile, Code *c);

void flush_codes(int outfile);

stack.c:
This file will contain the stack ADT implementation to be used by the decoder to reconstruct a Huffman tree

```
struct Stack {
    uint32_t top;
    uint32_t capacity;
    Node **items;
};
```

similar to previously implemented stacks but will store nodes instead
huffman.c:
This file will contain the implementation of the Huffman coding module which will be used by the encoder and decoder
Node *build_tree(uint64_t hist[static ALPHABET]);

void build_codes(Node *root, Code table[static ALPHABET]);

Node *rebuild_tree(uint16_t nbytes, uint8_t tree[static nbytes]);

void delete_tree(Node **root);

All pictures/screenshots taken from Professor Long's asgn6-v2-1.pdf