```c
#include "huffman.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

/*
Para definir o fim de um arquivo, podemos usar 3 estrat�gias:
1- usar um dos s�mbolos do alfabeto (mas precisamos nos certificar que este s�mbolo n�o apare�a
no texto)
2- usar um s�mbolo n+1 (sendo n o n�mero de s�mbolos do alfabeto)
3- contar a quantidade de caracteres durante a compress�o

To generate the object file: gcc -c huffman.c
*/

#define USE_SYMBOL_COUNT

////////////////////////////////////////////////////////////////////////
// Nos para o criacao da arvore
////////////////////////////////////////////////////////////////////////
ArvNo* arv_criano (char info, int freq, ArvNo* esq, ArvNo* dir) {
  ArvNo* p = (ArvNo*)malloc(sizeof(ArvNo));
  if (p != NULL) {
    p->info = info;
    p->freq = freq;
    p->esq = esq;
    p->dir = dir;
  }
  return p;
}

void arv_libera (ArvNo* r) {
  if (r == NULL) return;
  arv_libera(r->esq);
  arv_libera(r->dir);
  free(r);
}

void arv_imprime (ArvNo* r)
{
  printf("(");
  if (r != NULL) {
    printf("[%c,%d]", r->info, r->freq);
    arv_imprime(r->esq);
    printf(", ");
    arv_imprime(r->dir);
  }
  printf(")");
}

////////////////////////////////////////////////////////////////////////
// Frequencias
////////////////////////////////////////////////////////////////////////
HuffFrequencias* freq_cria (const char* arquivo_de_entrada) { //, char eof_simbolo) {
  // inicializa frequencias em 0
  int n_frequencias = NUMERO_MAXIMO_DE_SIMBOLOS;
  int frequencias[NUMERO_MAXIMO_DE_SIMBOLOS];
  memset(frequencias, 0, sizeof(int) * n_frequencias);

  // abre arquivo
  FILE *fp;
  fp = fopen(arquivo_de_entrada,"rb");
  if (fp == NULL) {
     printf("Erro ao tentar abrir arquivo!");
     exit(1);
  }
  fseek(fp, 0, SEEK_END); // mover o cursor para o final do arquivo
```

```
    long int fptamanho  = ftell(fp); // tamanho do arquivo em bytes
    printf("Tamanho do arquivo: %d bytes\n", fptamanho);

    // calcula frequencias de cada caractere
    fseek(fp, 0, SEEK_SET); // mover o cursor para o inicio do arquivo
    int c = 0;
    do {
      c = fgetc(fp); // ler um caractere
      if (c != EOF) { // se c for diferente de "End Of File"
        frequencias[c]++; // adicionar frequencia
      }
    } while (c != EOF);
    // Aqui, usaremos uma "gambiarra" tratando o EOF como '\0'
    // '\0' ◆ associado ao numero 0
  #ifndef USE_SYMBOL_COUNT
    frequencias[eof_simbolo]++;
  #endif

    fclose(fp);

    // conta numero de caracteres diferentes que apareceram
    int num_de_caracteres = 0;
    for (int i = 0; i < n_frequencias; i++) {
      if (frequencias[i] > 0){
        num_de_caracteres++;
      }
    }

    // cria vetor de frequencias para o heap
    HuffFrequencias* freq = (HuffFrequencias*)malloc(sizeof(HuffFrequencias));
    freq->f = (int*)malloc(sizeof(int) * num_de_caracteres);
    freq->c = (char*)malloc(sizeof(char) * num_de_caracteres);
    freq->n = num_de_caracteres;
    int pri = 0;
    for (int i = 0; i < n_frequencias; i++) {
      if (frequencias[i] > 0) {
        freq->f[pri] = frequencias[i];
        freq->c[pri] = (char)i;
        pri++;
      }
    }

    return freq;
  }

  int freq_tamanho (HuffFrequencias* f)
  {
    return f->n;
  }

  char freq_caractere (HuffFrequencias* f, int i)
  {
    return f->c[i];
  }

  int freq_frequencia (HuffFrequencias* f, int i)
  {
    return f->f[i];
  }

  void freq_imprime (HuffFrequencias* f)
  {
    printf("Numero de caracteres: %d\n", f->n);
    for (int i = 0; i < f->n; i++) {
      if (f->c[i] == '\0') {
        printf("\\0: %d\n", f->f[i]);
      }
      else {
        printf("%c: %d\n", f->c[i], f->f[i]);
```

```
    }
  }
}

void freq_libera (HuffFrequencias* f)
{
  free(f->f);
  free(f->c);
  free(f);
}

//////////////////////////////////////////////////////////////////////
// Árvore
//////////////////////////////////////////////////////////////////////
HuffArvore* arvh_cria (ArvNo* r)
{
  HuffArvore* h = (HuffArvore*)malloc(sizeof(HuffArvore));
  h->raiz = r;
  return h;
}

void arvh_imprime (HuffArvore* a)
{
  arv_imprime(a->raiz);
  printf("\n");
}

void arvh_libera (HuffArvore* a)
{
  arv_libera(a->raiz);
  free(a);
}

//////////////////////////////////////////////////////////////////////
// Dicionário
//////////////////////////////////////////////////////////////////////
static void dict_popula (ArvNo* r, HuffDicionario* dict, unsigned int bitcode, unsigned int
bitsize) {
  ArvNo* e = arvno_esq(r);
  ArvNo* d = arvno_dir(r);
  if (e == NULL && d == NULL) {
    char c = arvno_caractere(r);
    bool found = false;
    for (int i = 0; i < dict->n; i++) {
      if (c == dict->simbolos[i].simbolo) {
        found = true;
        if (dict->simbolos[i].tamanho > 0) {
          printf("Erro ao popular dicionario\n");
          exit(1);
        }
        dict->simbolos[i].codigo = bitcode;
        dict->simbolos[i].tamanho = bitsize;
      }
    }
    if (!found) {
      printf("Erro ao popular dicionario\n");
      exit(1);
    }
    return;
  }

  bitsize++;
  bitcode = bitcode << 1;
  dict_popula(e, dict, bitcode, bitsize);
  bitcode |= 1; // adiciona bit direito
  dict_popula(d, dict, bitcode, bitsize);
}

HuffDicionario* dict_cria (HuffFrequencias* f, HuffArvore* a)
```

```
{
  HuffDicionario* dict = (HuffDicionario*)malloc(sizeof(HuffDicionario));
  dict->n = freq_tamanho(f);
  dict->simbolos = (HuffSimbolo*)malloc(sizeof(HuffSimbolo)*dict->n);
  for (int i = 0; i < dict->n; i++) {
    char c = freq_caractere(f, i);
    dict->simbolos[i].simbolo = c;
    dict->simbolos[i].codigo = 0;
    dict->simbolos[i].tamanho = 0;
  }

  unsigned int bitcode = 0;
  unsigned int bitsize = 0;
  dict_popula(a->raiz, dict, bitcode, bitsize);

  return dict;
}

void dict_imprime (HuffDicionario* d)
{
  // imprime dicionario de simbolos
  for (int i = 0; i < d->n; i++) {
    printf("%d(%c) = ", i, d->simbolos[i].simbolo);
    for (int b = d->simbolos[i].tamanho - 1; b >= 0; b--) {
      if ((1 << b) & (d->simbolos[i].codigo)) {
        printf("1");
      }
      else {
        printf("0");
      }
    }
    printf("\n");
  }
}


void dict_libera (HuffDicionario* d)
{
  free(d->simbolos);
  free(d);
}

//////////////////////////////////////////////////////////////////////
// Compress�o e Descompress�o
//////////////////////////////////////////////////////////////////////
int huffman_comprime (HuffDicionario* hd, const char* arquivo_de_entrada, const char*
arquivo_de_saida)
{
  // abre arquivo de entrada
  FILE *fp;
  fp = fopen(arquivo_de_entrada,"rb");
  if (fp == NULL) {
     printf("Erro ao tentar abrir arquivo!");
     exit(1);
  }
  // mover o cursor novamente para o inicio do arquivo
  fseek(fp, 0, SEEK_SET);

  int number_of_encoded_symbols = 0;

  // abre arquivo de sa�da (comprimido)
  FILE *wp;
  wp = fopen(arquivo_de_saida,"wb");
  if (wp == NULL) {
     printf("Erro ao tentar abrir arquivo!");
     exit(1);
  }

  char output_byte = 0;
```

```
    int output_bitslivres = 8;
    int comprimindo = 1;
    do {
      // l□ um caractere
      char c = fgetc(fp);
      char ascii_code = c;
      // se for eof, seta o caractere como \0 (gambiarra utilizada >:) )
      if (c == EOF) {
        ascii_code = (char)0;
        comprimindo = 0;
  #ifdef USE_SYMBOL_COUNT
        break;
  #endif
      }

      // n□o □ EOF, conta o novo s□mbolo que ser□ escrito no arquivo comprimido
      number_of_encoded_symbols++;

      // busca o c□digo, em bits, do caractere lido
      unsigned int data_bits = 0;
      int data_nbits = 0;
      for(int i = 0; i < hd->n; i++) {
        if (hd->simbolos[i].simbolo == ascii_code) {
          data_bits = hd->simbolos[i].codigo;
          data_nbits = hd->simbolos[i].tamanho;
          break;
        }
      }

      // escreve os bits no arquivo comprimido
      while (data_nbits > output_bitslivres) {
        // bits restantes que v□o ficar para pr□xima escrita
        int bits_restantes = data_nbits - output_bitslivres;
        // calcula sobra
        unsigned int bits_sobra = data_bits % (1 << bits_restantes);

        // diminui o restante do dado que ser□ escrito depois
        unsigned int bits_para_adicao = data_bits - bits_sobra;
        output_byte += (char)(bits_para_adicao >> bits_restantes);

        data_bits = bits_sobra;
        data_nbits = bits_restantes;

        fputc(output_byte, wp);
        output_byte = 0;
        output_bitslivres = 8;
      }
      // se a quantidade de bits para escrever □ menor ou igual que os bits dispon□veis
  (output_bitslivres - data_nbits >= 0)
      output_byte += (data_bits << (output_bitslivres - data_nbits));
      output_bitslivres -= data_nbits;
      if (output_bitslivres == 0) {
        fputc(output_byte, wp);
        output_byte = 0;
        output_bitslivres = 8;
      }
    } while (comprimindo == 1);
    if (output_bitslivres < 8) { // se ultimos caracteres n□o completaram um byte
      fputc(output_byte, wp);
      output_byte = 0;
      output_bitslivres = 8;
    }
    fclose(wp);
    fclose(fp);
    return number_of_encoded_symbols;
  }

  void huffman_descomprime (HuffArvore* ha, const char* arquivo_de_entrada, const char*
  arquivo_de_saida, int numero_de_caracteres)
```

```
{
  int n = numero_de_caracteres;
  // abre arquivo de entrada
  FILE *fp;
  fp = fopen(arquivo_de_entrada,"rb");
  if (fp == NULL) {
     printf("Erro ao tentar abrir arquivo!");
     exit(1);
  }
  // mover o cursor novamente para o inicio do arquivo
  fseek(fp, 0, SEEK_SET);

  // abre arquivo de sa�da (comprimido)
  FILE *wp;
  wp = fopen(arquivo_de_saida,"wb");
  if (wp == NULL) {
     printf("Erro ao tentar abrir arquivo!");
     exit(1);
  }

  ArvNo* decode_no = ha->raiz;
  ArvNo* e = arvno_esq(decode_no);
  ArvNo* d = arvno_dir(decode_no);
  if (e == NULL && d == NULL) {
    printf("Arquivo vazio, apenas eof gravado.");
  }
  else {
    //int decodificando_arquivo = 1;
    char c = 0;
    int bit_pos = 0;
    do {
      if (bit_pos == 0) {
        c = fgetc(fp);
        bit_pos = 8;
      }
      if ((1 << (bit_pos - 1)) & c) { // se o bit estiver ativo
        decode_no = arvno_dir(decode_no);
      }
      else {
        decode_no = arvno_esq(decode_no);
      }
      e = arvno_esq(decode_no);
      d = arvno_dir(decode_no);
      if (e == NULL && d == NULL) {
        char v = arvno_caractere(decode_no);
        //if (v == eof_simbolo) {
        //  decodificando_arquivo = 0;
        //  //printf("\n");
        //} else {
          //printf("%c", v);
          fputc(v, wp);
          decode_no = ha->raiz;
        //}
        n--;
      }
      bit_pos--;
    } while (n > 0);//  decodificando_arquivo == 1);
  }
  fclose(wp);
  fclose(fp);
}
```