Mateusz Pepla

WCY22KY2S1

Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego

Zadanie: Implementacja kodera/dekodera słownikowego. Metoda: LZ77.

Analiza:

- Metoda LZ77 składa się z kroku początkowego, który polega na inicjalizacji słownika, wypełnieniu bufora oraz zapisaniu pierwszego słowa zakodowanego; oraz cyklicznego kroku kodowania polegającego na znajdowaniu najdłuższej sekwencji bufora powtarzającej się w słowniku, zapisaniu słowa zakodowanego oraz przesunięciu symboli o odpowiednią ilość miejsc (wejście -> bufor -> słownik). Słowa zakodowane składa się z trzech informacji: pozycji pierwszego symbolu sekwencji w słowniku, długości sekwencji, pierwszego znaku znajdującego się za sekwencją w buforze. Implementacja wymaga zatem funkcji: inicjującej algorytm, szukającej sekwencji, przesuwającej symbole oraz zapisującej słowo zakodowane.

- Dekodowanie opiera się na inicjacji słownika, a następnie na odczytywaniu sekwencji symboli na podstawie słów zakodowanych i przesuwaniu symboli (odczytana sekwencja -> słownik). Do implementacji dekodera potrzebne będą funkcje: inicjująca słownik, odczytująca słowa zakodowane z pliku, przesuwająca symbole.

Założenia:

W trakcie realizacji zadania przyjąłem następujące założenia: wielkość bufora jest mniejsza lub równa wielkości słownika; maksymalna sekwencja ma wielkość równa buforowi – 1, tak aby podczas przesuwania symboli każdy symbol przeszedł przez bufor; algorytm kodowania kończy działanie, gdy na pierwszym miejscu bufora znajduje się symbol EOF.

Algorytm:

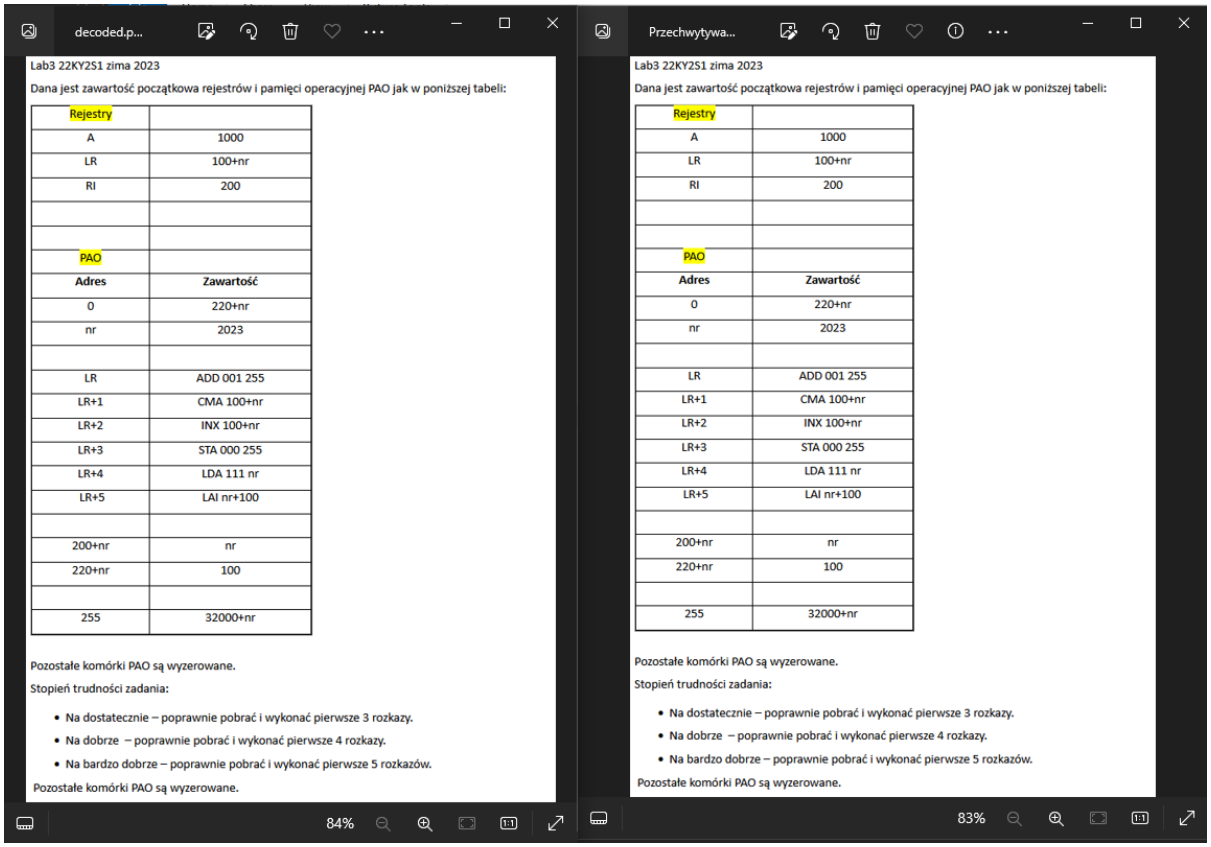
- Funkcja seek dictionary odnajduje sekwencje. Początkowo zakłada jej długość bk = 0. Następnie przy użyciu dwóch pętli porównuje sekwencje z bufora i słownika. Jeśli znajdzie dłuższą niż obecne bk, to je nadpisuje i zapisuje słowo zakodowane. Pętle kończą się po przejściu przez cały słownik. Jeśli bk nadal jest równe zero, zapisywane jest słowo kodowe = (0, 0, pierwszy symbol bufora). Jeśli pierwszy symbol za sekwencją to EOF, to zmniejszamy wielkość sekwencji o 1, tak aby EOF jest został przeniesiony do słownika.
- Funkcja move przemieszcza o l miejsc symbole w słowniku, z bufora do słownika, w buforze, z wejścia do słownika.
- Funkcja initialize odczytuje symbol z wejścia wypełnia nim słownik i wypełnia bufor kolejnymi symbolami z wejścia. Zapisuje pierwsze słowo zakodowane jako (0, 0, pierwszy symbol z wejścia).
- Funkcja move decoding dekoduje sekwencje symboli na podstawie pobrane słowa zakodowanego, a następnie przesuwa symbole w słowniku o długość sekwencji i na koniec słowniku wpisuje sekwencje.

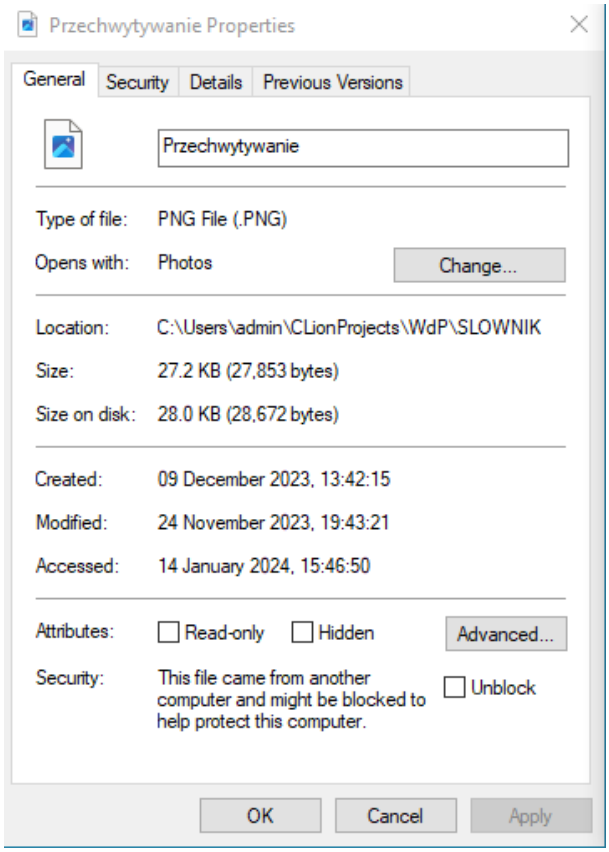
Testowanie:

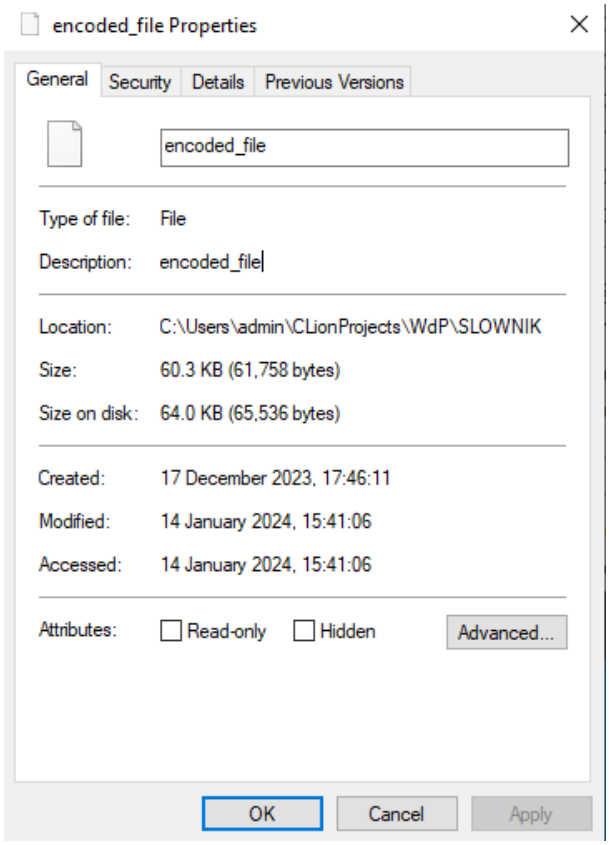
Plik zakodowany – encoded_file

Plik zdekodowany – decoded_file

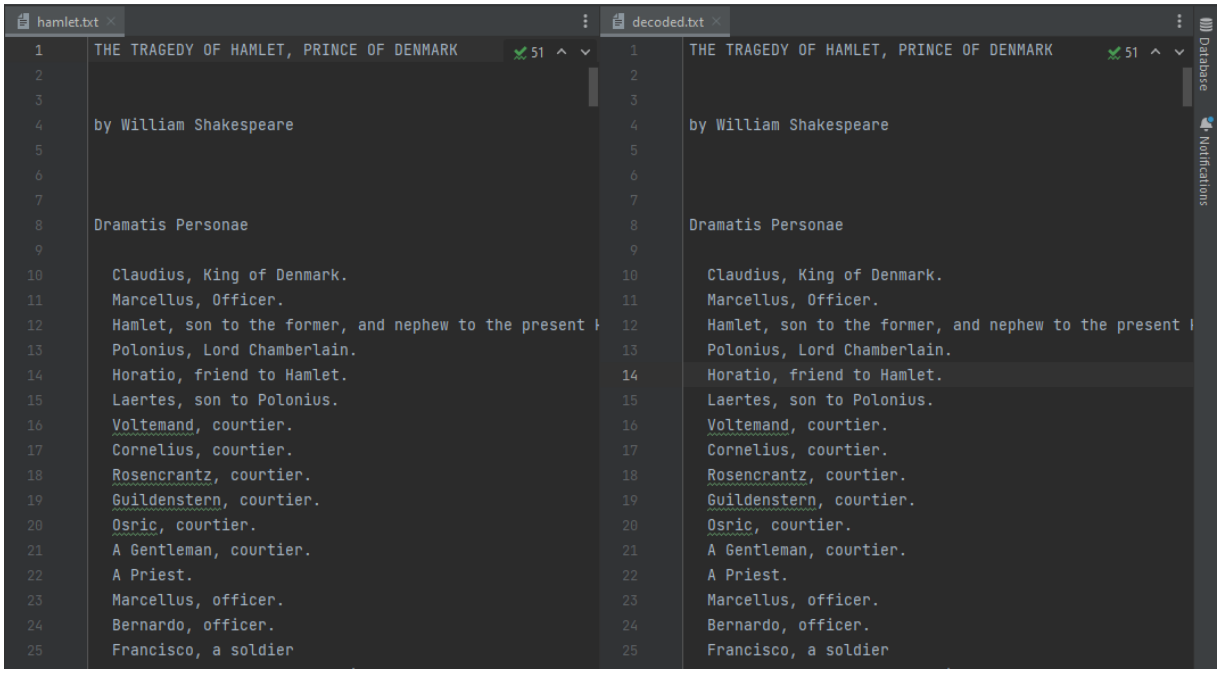
Testowanie na Przechwytywanie.PNG:

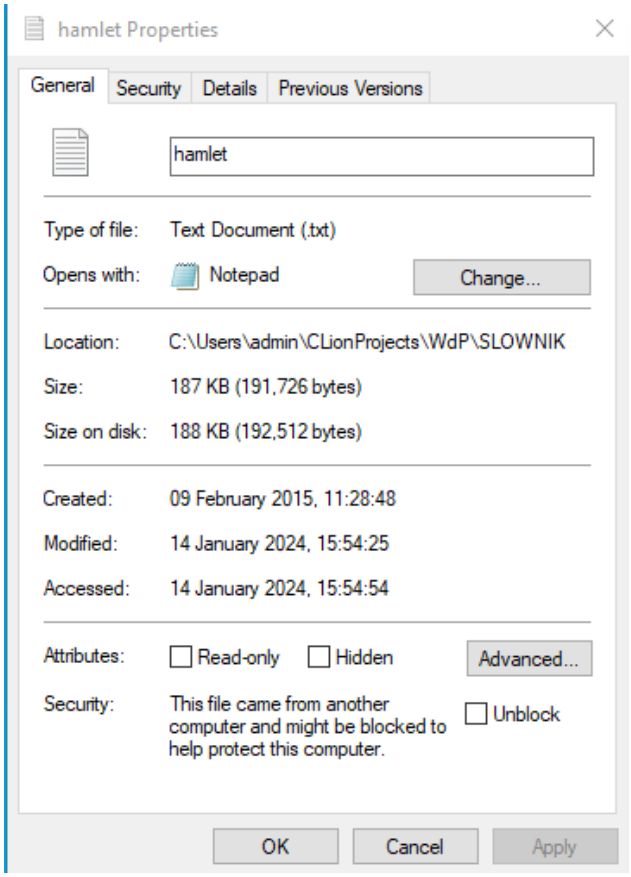


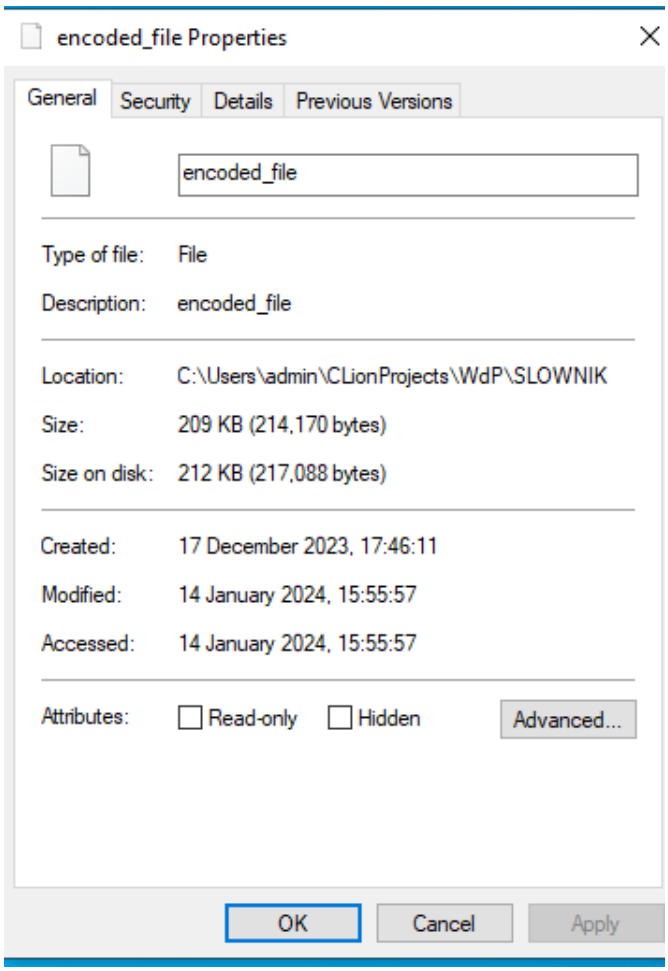




Testowanie na Przechwytywanie.PNG:


hamlet.txt
decoded.txt
THE TRAGEDY OF HAMLET, PRINCE OF DENMARK
by William Shakespeare
Dramatis Personae
Claudius, King of Denmark.
Marcellus, Officer.
Hamlet, son to the former, and nephew to the present
Polonius, Lord Chamberlain.
Horatio, friend to Hamlet.
Laertes, son to Polonius.
Voltemand, courtier.
Cornelius, courtier.
Rosencrantz, courtier.
Guildenstern, courtier.
Osric, courtier.
A Gentleman, courtier.
A Priest.
Marcellus, officer.
Bernardo, officer.
Francisco, a soldier
Database
Notifications


hamlet Properties
General
Security
Details
Previous Versions
hamlet
Type of file: Text Document (.txt)
Opens with: Notepad
Change...
Location: C:\Users\admin\CLionProjects\WdP\SLOWNIK
Size: 187 KB (191,726 bytes)
Size on disk: 188 KB (192,512 bytes)
Created: 09 February 2015, 11:28:48
Modified: 14 January 2024, 15:54:25
Accessed: 14 January 2024, 15:54:54
Attributes: Read-only Hidden
Advanced...
Security: This file came from another computer and might be blocked to help protect this computer.
Unblock
OK
Cancel
Apply



Wnioski:

Program działa poprawnie, to znaczy zakodowany i zdekodowany plik PNG i txt nie różnią się od oryginału. Kompresja dla pliku PNG nie zaszła. Zakodowany plik jest ponad dwa razy większy niż oryginał. Dla Plików TXT ich wielkości są zbliżone, plik zakodowany jest większy o 10%. Kompresje można by uzyskać zapisując słowa zakodowane w efektywniejszy sposób. Obecnie program zapisuje 3 short int.

Kod:

```
#include <stdio.h>

/* Encoder */

void seek_dictionary(int n, int m, int buffer[n], int dictionary[m], short int *p, short int *l, short int *c){
   /*
   *     This function looks for longest sequence of symbols
   *     from buffer that repeats itself in dictionary.
   *
   *     Input:
   *        Values n, m are lengths of buffer and dictionary respectively
   *        Buffer, Dictionary
   *
   *     Output:
   *        Results of a search are:
   *        p - position of first symbol of sequence in dictionary
   *        l - length of sequence
   *        c - first symbol after sequence in buffer.
   */

   int bk=0;                                  // length of longest found sequence
   for(int i=0; i+bk<m; i++){                        // loops through dictionary
      int tk=0;                                 // length of currently checked sequence
      for(int j=0; j<n-1; j++){                       // loops through buffer
         if(j+i >= m || buffer[j] != dictionary[j+i])         // checks for end of dictionary or if symbol
            break;                               // do not match
         else                                    // otherwise increases tk
            tk++;                                // and checks following symbol
```

```
        }

        if(tk > bk){                        // checks if new longest sequence is found
            bk = tk;                        // saves new longest length and assigns
            *p = (short int)i;                  // p, l ,c accordingly
            *l = (short int)bk;
            *c = (short int)buffer[bk];
        }
    }

    if(bk==0){                          // no symbol in dictionary matches
        *p = 0;                         // first symbol of buffer
        *l = 0;
        *c = (short int)buffer[0];
    }


    if(*c == EOF){                      // correction for the last output
        *l = (short int)(bk-1);             // value of p remains the same
        *c = (short int)buffer[bk-1];
    }
}

void move(int n, int m, int buffer[n], int dictionary[m], int l, FILE *in){
    /*
    *   This function shuffles symbols in buffer and dictionary, so
    *   first l symbols of dictionary are moved out of it,
    *   remaining symbols of dictionary are moved l spaces ahead, then
    *   first l symbols of buffer are moved int dictionary as last l symbols,
    *   remaining symbols of buffer are moved l spaces ahead, and the last l
    *   symbols of buffer are filled with symbols from input (in).
```

```
    *
    *     Input:
    *         l - number of spaces symbols are moved
    *         in - input file
    *
    *     Output:
    *         This function does not return anything, but result of operation is
    *         that buffer and dictionary are shuffled as described above.
    */

    for (int i = l; i < m; i++)                    // move symbols in dictionary l spaces ahead
        dictionary[i-l] = dictionary[i];
    for (int i = m-l; i < m; i++)                  // move l symbols from buffer to dictionary
        dictionary[i] = buffer[i-m+l];
    for (int i = l; i < n; i++)                    // move symbols in buffer l spaces ahead
        buffer[i-l] = buffer[i];
    for (int i = n-l; i < n; i++)                  // move symbols from input to dictionary
        buffer[i] = fgetc(in);
}

void initialize(int n, int m, int buffer[n], int dictionary[m], FILE *in, short int *p, short int *l, short int
*c){
    /*
    *     This function starts LZ77 encoding.
    *
    *     Input:
    *         Values n, m are lengths of buffer and dictionary respectively
    *         Buffer, Dictionary
    *         input file in
    *
    *     Output:
```

```
    *          c - first symbol from input (in)
    *          p = l = 0
    *          All symbols of dictionary are set to c.
    *          Buffer is filled with symbols from input.
    */

    *c = (short int) fgetc(in);                    // C is set to symbol from input
    for(int i=0; i<m; i++)                          // symbols of dictionary are set to c
       dictionary[i] = *c;
    for(int i=0; i<n; i++)                         // buffer filled with symbols from input
       buffer[i] = fgetc(in);
    *p = 0;
    *l = 0;
}

void write_encoded_element(FILE *out, short int *p, short int *l, short int *c){
    /*
    *     This function writes encoded information into output file out.
    *
    *     Input:
    *          p - position of first symbol of sequence in dictionary
    *          l - length of sequence
    *          c - first symbol after sequence in buffer.
    *
    *     Output:
    *          Values of p, l and c are written into output file out as 2 bytes each.
    */

    fwrite((const void*) p, sizeof(short int), 1, out);
    fwrite((const void*) l, sizeof(short int), 1, out);
    fwrite((const void*) c, sizeof(short int), 1, out);
```

```
}

void encoder(int n, int m){
   /*
   *     This function encodes file with LZ77 encoding.
   *
   *     Input:
   *         Values n, m are lengths of buffer and dictionary respectively
   *         input file in
   *
   *     Output:
   *         This function does not return anything, but the result of it
   *         is encoded file.
   */

   // Declaration of values and tables
   char *file_name="C:\\Users\\admin\\CLionProjects\\WdP\\SLOWNIK\\hamlet.txt";
   char *file_name_out="C:\\Users\\admin\\CLionProjects\\WdP\\SLOWNIK\\encoded_file";
   FILE *in, *out;                                  // in - file, out - encoded file
   short int p, l, c;                               // position, length, symbol (see seek_dictionary)
   int buffer[n];
   int dictionary[m];

   in = fopen(file_name, "rb+");                    // opens file binary in reading mode
   out = fopen(file_name_out, "wb+");               // opens file binary in writing mode

   if(in == NULL || out == NULL)                          // checks for io errors
      fputs("error: opening file failed", stdout);
   else{
      initialize(n, m, buffer, dictionary, in, &p, &l, &c);
      write_encoded_element(out, &p, &l, &c);
```

```
        // seeks sequence and writes encoded information, then repeats process until end of file
        while (1){
            seek_dictionary(n, m, buffer, dictionary, &p, &l, &c);
            write_encoded_element(out, &p, &l, &c);
            move(n, m, buffer, dictionary, l+1, in);

            if(buffer[0]==EOF)
                break;
        }

    }
    fclose(in);
    fclose(out);
}

/* Decoder */

void get_encoded_element(FILE *in, short int *p, short int *l, short int *c){
    /*
    *     This function gets encoded element from encoded file.
    *
    *     Input:
    *         in - encoded file
    *
    *     Output:
    *         p - position of first symbol of sequence in dictionary
    *         l - length of sequence
    *         c - first symbol after sequence in buffer.
    */
```

```
    fread(p, sizeof(short int), 1, in);
    fread(l, sizeof(short int), 1, in);
    fread(c, sizeof(short int), 1, in);
}

void initialize_decoding(int m, int dictionary[m], FILE *in, FILE *out, short int *p, short int *l, short int *c){
    /*
    *     This function starts decoding encoded file.
    *
    *     Input:
    *         First encoded element:
    *         p - position of first symbol of sequence in dictionary
    *         l - length of sequence
    *         c - first symbol after sequence in buffer.
    *
    *     Output:
    *         This function does not return anything, but the result of it
    *         is decoded sequence of symbols and dictionary filled with first symbol.
    *
    */


    get_encoded_element(in, p, l, c);                    // gets first encoded element
    for(int i=0; i<m; i++)                               // fills dictionary with first symbol
        dictionary[i] = *c;
    putc((char) *c, out);                                // loads decoded symbol into output
}

void move_decoding(int m, int dictionary[m], FILE *out, short int p, short int l, short int c){
    /*
```

```
    *     This function decodes single encoded element from encoded file.
    *
    *     Input:
    *        Encoded element:
    *        p - position of first symbol of sequence in dictionary
    *        l - length of sequence
    *        c - first symbol after sequence in buffer.
    *        Dictionary.
    *
    *     Output:
    *         This function does not return anything, but the result of it
    *         is decoded sequence of symbols and dictionary with symbols
    *         shuffled l+1 spaces ahead.
    *
    */
    int decoded_symbols[l+1];                          // sequence of decoded symbols (empty)

    for(int i = p; i < p+l; i++)                       // loads decoded symbols dictionary according
       decoded_symbols[i-p] = dictionary[i];                  // to p and l values
    decoded_symbols[l] = c;                           // last decoded symbol is c value from element

    for(int i=0; i<l+1; i++)                           // loads decoded sequence into output file
       putc((char) decoded_symbols[i], out);

    for (int i = l+1; i < m; i++)                       // shuffles symbols l+1 spaces ahead
       dictionary[i-l-1] = dictionary[i];

    for(int i=0; i<l+1; i++)                           // last symbols of dictionary are set to
       dictionary[m-l-1+i] = decoded_symbols[i];              // the decoded sequence

}
```

```
void decoder(int m){
   /*
   *     This function decodes file encoded with LZ77.
   *
   *     Input:
   *        Dncoded file.
   *
   *     Output:
   *         Decoded file.
   */

   char *file_name="C:\\Users\\admin\\CLionProjects\\WdP\\SLOWNIK\\encoded_file";
   char *file_name_out="C:\\Users\\admin\\CLionProjects\\WdP\\SLOWNIK\\decoded.txt";
   FILE *in, *out;
   short int p, l, c;                              // Saves values of encoded element
   int dictionary[m];                                 // position, length, symbol

   in = fopen(file_name, "rb+");                 // opens files in reading and writing mode
   out = fopen(file_name_out, "wb+");              // accordingly.

   initialize_decoding(m, dictionary, in, out, &p, &l, &c);// Starts process of decoding

   while(1){                                          // Gets encoded element from file
      get_encoded_element(in, &p, &l, &c);             // and then decodes it using
                                                 // move_decoding function
      if(feof(in))                                  // repeats the process until end of file
         break;

      move_decoding(m,dictionary, out, p, l, c);
   }
```

```
    fclose(in);                                 // closes files
    fclose(out);
}

int main() {
    encoder(250, 5000);
    decoder(5000);

    return 0;
}
```