Харківський університет радіоелектроніки
Факультет комп'ютерних наук
Кафедра програмної інженерії

ЗВІТ

до практичного заняття №2 з дисципліни
"Безпека програм та даних"
на тему "ЗНАЙОМСТВО З ШИФРОМ «ОДНОРАЗОВИЙ БЛОКНОТ»"

Виконав ст. гр ПЗПІ-20-2
Овчаренко Михайло Миколайович

Перевірив
асистент кафедри ПІ
Олійник О. О.

Харків 2023

МЕТА

Ознайомити студентів з шифром «одноразовий блокнот», відпрацювати навички використовування цього шифру для кодування та декодування тексту.

ЗАВДАННЯ

Розробити програму для шифрування та дешифрування тексту в кодовій сторінці UTF-8 алгоритмом одноразового блокноту за допомогою таблиці стиснення, що наведено у методичці до цього ПЗ.

ХІД РОБОТИ

Оскільки кожний стиснений символ російської абетки являє собою пару цифр від 0 до 10, то не має сенсу виділяти під кожну цифру **один** байт. Замість того, у кожному байті міститься **дві** таких цифри, кожна займаючи 4 біта. Перетворивши ключ та текст на числову послідовність, просто робимо додавання за модулем 10 для шифрування та віднімання для дешифрування.

В багатьох ОС кодуванням файлів з похідним текстом програми є за замовченням UTF-8. З цього випливає, що строкові літерали, що зберігаються в сегменті даних виконуваного файлу, будуть транслюватися компілятором буквально як послідовність байтів у форматі UTF-8. І тут ми зтикаємося з ситуацією, коли ми не можемо, просто **віднявши** від одного ASCII-байту літеру ‘А’, отримати його порядковий номер в абетці, тому що:

1) ми працюємо не з літерами простору Basic Latin, а з кирилицею, а вона представляється двома байтами замість одного;

2) навіть якщо в програмі потрібно працювати лише з латиницею (займає один байт), але у форматі UTF, то гарною практикою буде працювати з таким текстом саме як з послідовністю UTF, із міркувань розширюваності функціоналу програми та сумісності з будь-якими абетками, що існують.

З цією метою я використовував три функції (рис. 1).

```
20 int utf8_codepoint_length(const uint8_t byte);
21 void utf8_encode(uint32_t codepoint, uint8_t utf8_sequence[4], int* length);
22 uint32_t utf8_decode(const uint8_t* sequence, int length);
```

Рисунок 1 – Функції для обробки UTF-8

Перша обчислює довжину послідовності; третя – вираховує так звану «кодову точку», що фактично є порядковим номером символу стандарту Unicode, з байтової послідовості; друга – протилежна третій. Те, як влаштований UTF-8 можемо побачити на рис. 2.

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Рисунок 2 – Принцип кодування UTF-8

Саме ці функції я використовував при побудові алгоритму, працюючи з символами. Імплементацію функцій можна побачити в файлі «utf.c» з додатку А, а

сама програма з усіма функціями роботи з таблицею стиснення та шифрування одноразовим блокнотом, наведені в файлі «main.c».

Продемонструємо роботу програми (див. рис. 3).

```
michael@DESKTOP-ERVQ9EV:~/Stud/Labs/CryptographyLabs/PZ2$ make run
./app
Original text:
1 81 82 83 84 4
АБВГДЕ
Encrypted text:
9 25 37 61 28 9
Decrypted text:
1 81 82 83 84 4
АБВГДЕ
michael@DESKTOP-ERVQ9EV:~/Stud/Labs/CryptographyLabs/PZ2$
```

Рисунок 3 – Початковий текст співпадає з розшифрованим

Отже, алгоритм алгоритм реалізовано правильно.

ВИСНОВКИ

Розробив програму мовою C для шифрування та дешифрування Unicode тексту в кодовій сторінці UTF-8 алгоритмом одноразового блокноту.

ДОДАТОК А

Програмний код

Файл utf.c

```
 1  #include <stdint.h>
 2
 3  int utf8_codepoint_length(const uint8_t byte) {
 4      if ((byte & 0x80) == 0) {
 5          return 1; // Однобайтовая кодовая точка
 6      } else if ((byte & 0xE0) == 0xC0) {
 7          return 2; // Двухбайтовая кодовая точка
 8      } else if ((byte & 0xF0) == 0xE0) {
 9          return 3; // Трехбайтовая кодовая точка
10      } else if ((byte & 0xF8) == 0xF0) {
11          return 4; // Четырехбайтовая кодовая точка
12      } else {
13          return -1; // Недопустимая последовательность
14      }
15  }
16
17  void utf8_encode(uint32_t codepoint, uint8_t utf8_sequence[4], int* length)
{
18      if (codepoint <= 0x7F) {
19          // Single-byte character
20          utf8_sequence[0] = (uint8_t)codepoint;
21          *length = 1;
22      } else if (codepoint <= 0x7FF) {
23          // Two-byte character
24          utf8_sequence[0] = (uint8_t)((codepoint >> 6) | 0xC0);
25          utf8_sequence[1] = (uint8_t)((codepoint & 0x3F) | 0x80);
26          *length = 2;
27      } else if (codepoint <= 0xFFFF) {
28          // Three-byte character
```

```
29          utf8_sequence[0] = (uint8_t)((codepoint >> 12) | 0xE0);
30          utf8_sequence[1] = (uint8_t)(((codepoint >> 6) & 0x3F) | 0x80);
31          utf8_sequence[2] = (uint8_t)((codepoint & 0x3F) | 0x80);
32          *length = 3;
33      } else if (codepoint <= 0x10FFFF) {
34          // Four-byte character
35          utf8_sequence[0] = (uint8_t)((codepoint >> 18) | 0xF0);
36          utf8_sequence[1] = (uint8_t)(((codepoint >> 12) & 0x3F) | 0x80);
37          utf8_sequence[2] = (uint8_t)(((codepoint >> 6) & 0x3F) | 0x80);
38          utf8_sequence[3] = (uint8_t)((codepoint & 0x3F) | 0x80);
39          *length = 4;
40      } else {
41          // Invalid code point
42          *length = 0;
43      }
44  }
45
46  uint32_t utf8_decode(const uint8_t* sequence, int length) {
47      if (length <= 0) {
48          return 0;
49      }
50      uint32_t codepoint = 0;
51      if (length == 1) {
52          // Однобайтовая кодовая точка
53          codepoint = sequence[0];
54      } else {
55          // Многобайтовая кодовая точка
56          codepoint = sequence[0] & (0xFF >> (length + 1));
57          for (int i = 1; i < length; i++) {
58              codepoint <<= 6;
59              codepoint |= (sequence[i] & 0x3F);
60          }
61      }
62      return codepoint;
63  }
```

Файл main.c

```
 1  #include <stdio.h>
 2  #include <ctype.h>
 3  #include <stdlib.h>
 4  #include <stdint.h>
 5  #include <string.h>
 6
 7  #define A 0x410
 8
 9  // Assuming straddling table contains only russian letters,
10  // in UTF-8 encoding they are represented by two bytes
11
12                                                          //  1234567...
column id
13  char* straddling_utf8 = "АИТЕСНО"
14  "БВГДЖЗКЛМП" // 8 row id
15  "РУФХЦЧШЩЪЫ" // 9
16  "ЬЭЮЯ"; // 0
17
18  int RL[4] = {7, 10, 10, 4}; // Row length in letters
19
20  int utf8_codepoint_length(const uint8_t byte);
21  void utf8_encode(uint32_t codepoint, uint8_t utf8_sequence[4], int*
length);
22  uint32_t utf8_decode(const uint8_t* sequence, int length);
23
24  int str_utf8_len(const char* str, int byte_len)
25  {
26          int char_count = 0;
27          for(int i = 0; i < byte_len; )
28          {
29                  int cplen = utf8_codepoint_length(str[i]);
30                  char_count++;
31                  i += cplen;
32          }
33          return char_count;
```

```
    34  }
    35
    36  uint32_t* straddling_codepoints()
    37  {
    38          int cp_count = str_utf8_len(straddling_utf8,
strlen(straddling_utf8));
    39          uint32_t* buf = (uint32_t*) calloc(cp_count, 4);
    40          int j = 0;
    41          for(int i = 0; i < strlen(straddling_utf8); )
    42          {
    43                  int cplen = utf8_codepoint_length(straddling_utf8[i]);
    44                  buf[j] = utf8_decode(straddling_utf8 + i, cplen);
    45                  i += cplen;
    46                  j++;
    47          }
    48          return buf;
    49  }
    50
    51  int8_t* compress_utf8_chars_to_nums(const char* str_utf8, int len)
    52  {
    53          int letter_count = str_utf8_len(str_utf8, len);
    54          int8_t* num_buf = (int8_t*) malloc(letter_count); // maximum
letter_count bytes
    55          memset(num_buf, 0, letter_count);
// one byte contains 1 or 2 letters (4 bits per digit)
    56          uint32_t* straddling_cps = straddling_codepoints();
    57          int straddling_len = str_utf8_len(straddling_utf8,
strlen(straddling_utf8));
    58          int j = 0, shl = 0;
    59
    60          for(int i = 0; i < len;)
    61          {
    62                  int cplen = utf8_codepoint_length(str_utf8[i]);
    63                  if(cplen > 0)
    64                  {
    65                          uint32_t cp = utf8_decode(str_utf8 + i, cplen);
    66                          int pos = -1;
```

```
    67                              for(int k = 0; k < straddling_len; k++)
    68                                      if(straddling_cps[k] == cp)
    69                                      {
    70                                              pos = k; break;
    71                                      }
    72
    73                              if(pos >= 0)
    74                              {
    75                                      // Calculate row and col
    76                                      int rlc = RL[0];
    77                                      char row = 0, col = 0;
    78                                      if(pos < rlc) {
    79                                              row = 0xF;
    80                                              col = pos + 1;
    81                                      }
    82                                      else {
    83                                              for(int l = 1; l < sizeof(RL) /
sizeof(RL[i]); l++)
    84                                              {
    85                                                      if(pos < rlc + RL[l])
    86                                                      {
    87                                                              row = (7 + l) % 10;
    88                                                              col = pos - rlc +
1;
    89                                                              break;
    90                                                      }
    91                                                      rlc += RL[i];
    92                                              }
    93                                      }
    94
    95                                      uint8_t result = (row << 4) | col;
    96                                      num_buf[j] = result;
    97                                      j++;
    98                              }
    99                              i += cplen;
   100                     }
```

```
101         }
102         free(straddling_cps);
103         return num_buf;
104 }
105
106 void one_time_pad(int8_t* src, int8_t* key, int len, char mode)
107 {
108         for(int i = 0; i < len; i++)
109         {
110                 char src_h = (src[i] >> 4) & 0x0F;
111                 char src_l = src[i] & 0x0F;
112                 char key_h = (key[i] >> 4) & 0x0F;
113                 char key_l = key[i] & 0x0F;
114                 // Don't cipher with key's 4h
115                 if(key_h & 0xF == 0xF) key_h = key_l;
116
117                 if(mode == 'e')
118                 {
119                         // Don't encrypt 4h part of src
120                         if(src_h != 0xF) src_h = (src_h + key_h) % 10;
121                         src_l = (src_l + key_l) % 10;
122                 }
123                 else if(mode == 'd')
124                 {
125                         if(src_h != 0xF) src_h = (src_h - key_h + 10) % 10;
126                         src_l = (src_l - key_l + 10) % 10;
127                 }
128                 src[i] = (src_h << 4) | (src_l & 0x0F);
129         }
130 }
131
132 void print_hex(int8_t* nt1, int t1_len)
133 {
134         for(int i = 0; i < t1_len; i++)
135         {
```

```
136                     if((int8_t) (nt1[i] & 0xF0) == (int8_t) 0xF0)
137                             printf("%X ", (int32_t)nt1[i] & 0x0000000F);
138                     else
139                             printf("%X ", (int32_t)nt1[i] & 0x000000FF);
140             }
141             puts("");
142     }
143
144     char* decompress_to_utf8_char(int8_t* nt, int nt_len)
145     {
146             char* str = (char*) malloc(nt_len * 2 + 1); // 2 bytes per sequence
+ ending
147             int j = 0;
148             for(int i = 0; i < nt_len; i++)
149             {
150                     if((nt[i] & 0xF0) == 0xF0)
151                     {
152                             char cpos = (nt[i] & 0x0F) - 1;
153                             // 1 ==> 1*2; 2 ==> 2*2 = 4
154                             cpos *= 2;
155                             str[j] = straddling_utf8[cpos];
156                             str[j + 1] = straddling_utf8[cpos + 1];
157                             j += 2;
158                     }
159                     else
160                     {
161                             char row = (nt[i] >> 4) & 0x0F;
162                             char col = (nt[i] & 0x0F) - 1;
163                             int cpos = 0;
164                             for(int k = 0; k < (row - 7 + 10) % 10; k++) // row
= 8 k < 1; sum of all prev rows lens
165                                     cpos += RL[k];
166                             cpos += col; // cpos = 7 col = 0 cpos = 7
167                             cpos *= 2; // 2 !!
168                             str[j] = straddling_utf8[cpos];
169                             str[j + 1] = straddling_utf8[cpos + 1];
```

```
170                         j += 2;
171                 }
172         }
173         str[nt_len * 2] = 0;
174         return str;
175 }
176
177
178 int main()
179 {
180         char* k1 = "ЛЕСЛЕС";
181         char* t1 = "АБВГДЕ";
182         int t1_len = str_utf8_len(t1, strlen(t1));
183
184         int8_t* nt1 = compress_utf8_chars_to_nums(t1, strlen(t1));
185         int8_t* nk1 = compress_utf8_chars_to_nums(k1, strlen(k1));
186         puts("Original text:");
187         print_hex(nt1, t1_len);
188         char* text1 = decompress_to_utf8_char(nt1, t1_len);
189         printf("%s\n", text1);
190
191         one_time_pad(nt1, nk1, t1_len, 'e');
192
193         puts("Encrypted text:");
194         print_hex(nt1, t1_len);
195         //char* text2 = decompress_to_utf8_char(nt1, t1_len);
196         //printf("%s\n", text2);
197
198         one_time_pad(nt1, nk1, t1_len, 'd');
199
200         puts("Decrypted text:");
201         print_hex(nt1, t1_len);
202         char* text3 = decompress_to_utf8_char(nt1, t1_len);
203         printf("%s\n", text3);
204
```

```
205           free(nt1);
206           free(nk1);
207           free(text1); free(text3);
208           exit(0);
209   }
210
```