

2016

Development guide for String Compression Project

מנחים: גב' מיקה עמית וגב' ליאת רוזנברג

עדי פרלוב, ת"ז: 200477842
רעות מזרחי, ת"ז: 305160079

המדריך נכתב לצוות פיתוח לצורך תחזוקה / הרחבה עתידית של הפרויקט.

האפליקציה נכתבה בשפת C++ תחת סביבת הפיתוח Microsoft visual studio 2015.

GUI – Microsoft windows forms.

המערכת בנויה מהמודולים הבאים:

LZ77 with mistakes Application

LZ77 with mistakes Algorithm

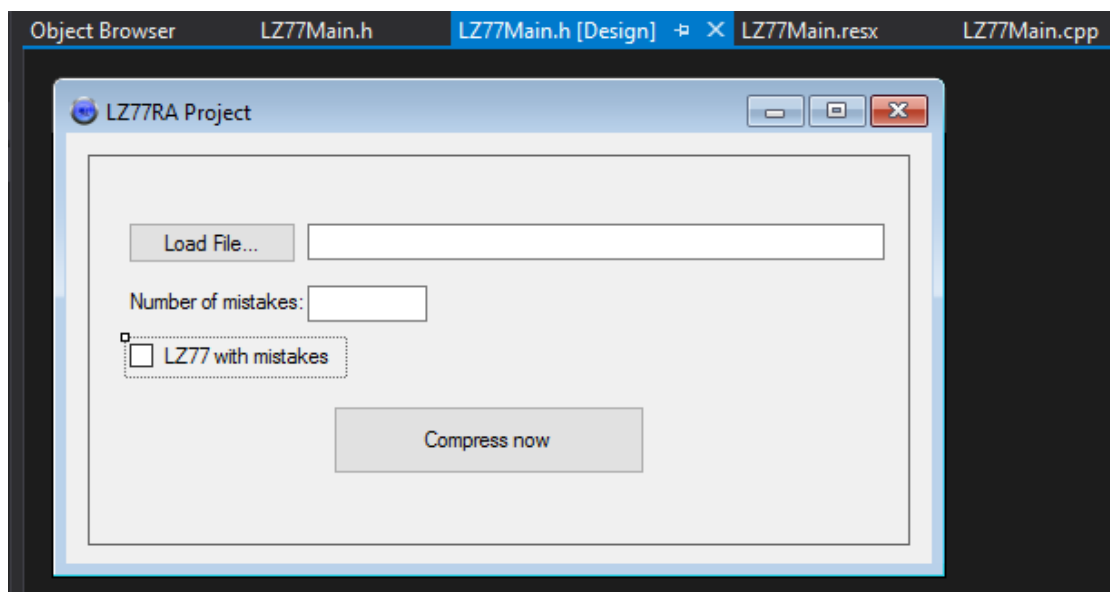
שלב ראשון:

LZ77 with mistakes Application : תחת ה-class הבא :

```
public ref class LZ77Main : public System::Windows::Forms::Form
```

נמצא את כל האובייקטים על וט, ניתן להרחיבם בקלות על ידי הוספת אובייקטים

חדשים מה- LZ77Main [Design]



המרת Text ל- Sting לצורך ניתוח המידע באלגוריתם:

האובייקטים ב- וט שמורים תחת type String^ ולכן יש צורך להשתמש באובייקט

```
#include <msclr\marshal_cppstd.h> msclr::interop::marshal_context
```

על מנת לשמור את הנתונים כי Std::String במימוש.

Dynamic Convert Decimal To Binary:

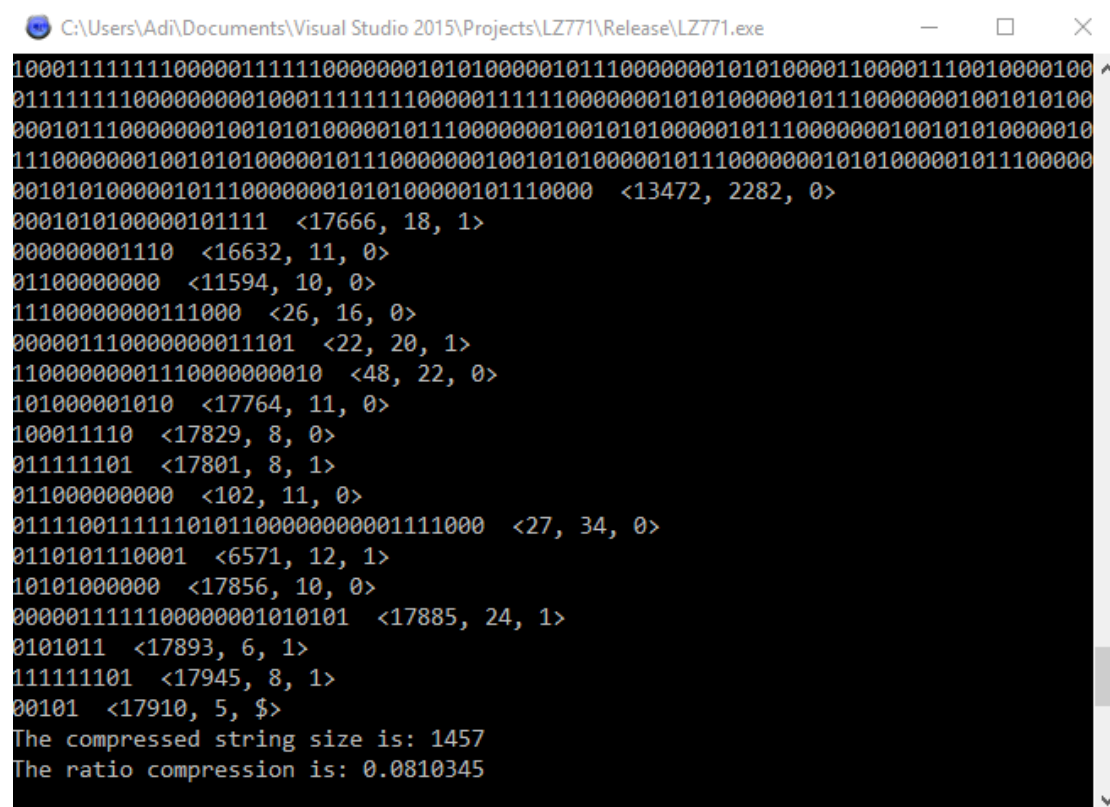
על מנת לבצע המרה של כל ה- words שמופיעים כמספרים עשרוניים. יש צורך לבצע המרה דינאמית לבינארי, מכיוון שאורך המחרוזת ידוע רק בזמן ריצה, לכן נשתמש באלגוריתם להמרה למספר בינארי:

```
unsigned long long int decimal_binary(int n) /* Function to convert decimal to binary.*/
{
    unsigned long long int rem, i = 1, binary = 0;

    while (n != 0)
    {
        rem = n % 2;
        n /= 2;
        binary += rem*i;
        i *= 10;
    }
    return binary;
}
```

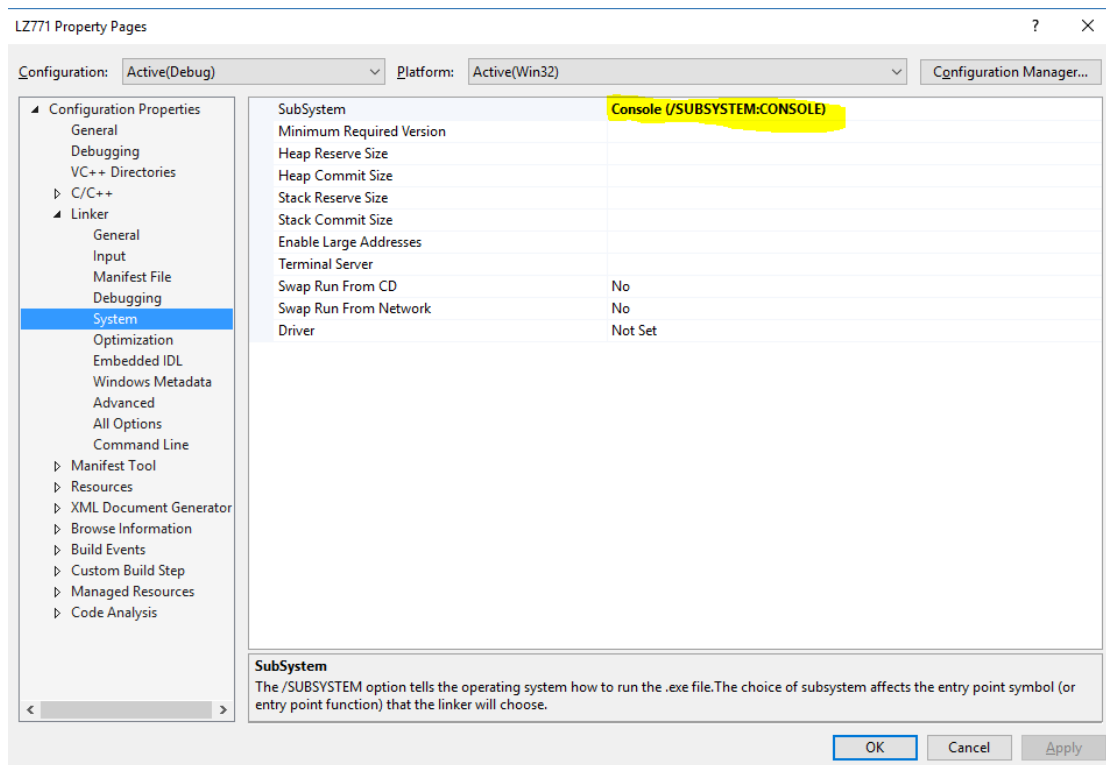
- אנו משתמשים ב- `long long int` על מנת לא לגרום לבעיית cyclic overflow של המשתנה i.
- בשלב הבא אנו מרפדים באפסים עד לגודל `log` של אורך המחרוזת ושומרים את המחרוזת במקום המתאים לפי ה- `word`.

Info Window:



```
C:\Users\Adi\Documents\Visual Studio 2015\Projects\LZ771\Release\LZ771.exe
10001111111100000111111000000010101000001011100000001010100001100001110010000100
0111111111000000000100011111110000011111000000010101000001011100000001001010100
00010111000000010010101000001011100000001001010100000101110000000100101010000010
11100000001001010100000101110000000100101010000010111000000010101000001011100000
0010101000001011100000001010100000101110000 <13472, 2282, 0>
000101010000010111 <17666, 18, 1>
000000001110 <16632, 11, 0>
01100000000 <11594, 10, 0>
11100000000111000 <26, 16, 0>
000001110000000011101 <22, 20, 1>
11000000001110000000010 <48, 22, 0>
101000001010 <17764, 11, 0>
100011110 <17829, 8, 0>
011111101 <17801, 8, 1>
011000000000 <102, 11, 0>
0111100111110101100000000001111000 <27, 34, 0>
0110101110001 <6571, 12, 1>
10101000000 <17856, 10, 0>
000001111100000001010101 <17885, 24, 1>
0101011 <17893, 6, 1>
111111101 <17945, 8, 1>
00101 <17910, 5, $>
The compressed string size is: 1457
The ratio compression is: 0.0810345
```

חלון האינפורמציה משמש לזיוזאליזציה של המידע המנותח מהאלגוריתם בלבד
והשליטה על הצגתו נעשית דרך ה- Linker->System



LZ77 with mistakes Algorithm:

חלק זה אחראי על ניתוח המחרוזת הנתונה, בחירת האינדקסים המתאימים לטעויות ובכך יצירת המחרוזת המכווצת.

שלב 1: טעינת המחרוזת נעשית ע"י `.std::ifstream`.

שלב 2: הלולאה המרכזית נמשכת כל עוד ה- `cursor` לא הגיע לסוף המחרוזת.

שלב 3: קריאה ל- `wrapper` לצורך זיהוי של אינדקסים כטעות עבור האיטרציה הנוכחית.

אם ה- `wrapper` זיהה אינדקס כטעות הוא משנה אותו ב- `buffer`, לכן הוא מועבר כ- `reference value`.

שלב 4: מציאת המחרוזת הארוכה ביותר על ה- `buffer` המעודכן.

שלב 5: יצירת `word`.

שלב 6: המרה של ה- `word` למספר בינארי.

שלב 7: הצגה של ה- `word & related string` ל- `info window`.

פעולת ה- `Wrapper`:

פעולת ה- `wrapper` נותנת לאלגוריתם את היכולת לזהות ביטים כ-טעות.

בתוך הלולאה הראשית עוברים על המחרוזת k פעמים (k הוא מספר הטעויות שניתן למצוא עבור כל איטרציה).

בכל איטרציה מוצאים את המחרוזת הארוכה ביותר בעזרת `.findLongestMatch()`.

מוצאים את מספר הפעמים שהמחרוזת מופיעה בהמשך ה- `Buffer` בעזרת `.forecast()`.

ועל פי הדירוג בוחרים את מספר הטעויות הרצוי.

משנים את ה- `buffer` על פי האינדקסים הרצויים כך שהאלגוריתם ירוץ על `buffer` מעודכן.