

## Projekt programistyczny nr 7- Tłumacz

Zmiany w treści zatwierdzone przez prowadzącego laboratoria:

- Zamiana komendy dla kierunku „L1>L2” na „L1-L2”, ponieważ konsola w systemie Windows wykorzystuje znak „>” w swoich poleceniach i koliduje to z wywoływaniem programu z konsoli.

Założenia przy tworzeniu programu:

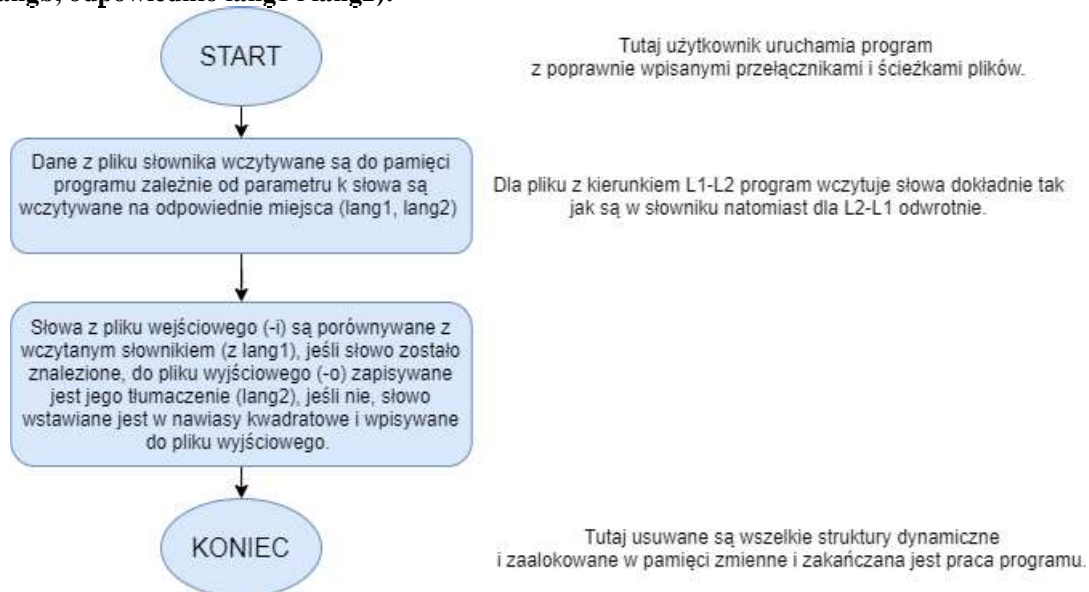
- Pliki słownika, wejściowe i wyjściowe muszą być w kodowaniu UTF-8 lub takim w którym znak nie przekracza 4 bajtów i muszą być plikami tekstowymi.
- Słowa w słowniku oddzielone są znakami spacji i końca linii
- Słowa składają się wyłącznie z liter, ich wielkość nie ma znaczenia.

Głównym zadaniem programu jest przetłumaczenie sposobem naiwnym słowa w jednym języku na drugi z wykorzystaniem pliku słownika który podaje użytkownik. Słowa nieprzetłumaczone zostają zapisane w nawiasach kwadratowych. Program operuje w całości na plikach w konsoli wyświetlane są tylko informacje i odpowiednie komunikaty. Słowa są przechowywane przez słownik w następującej postaci:

<słowo w języku1> <słowo w języku2>

Zdecydowałem się na zastosowanie struktur dynamicznych ze względu na to że nie jest znana nam wielkość pliku słownik a także słowa występują w różnych wielkościach.

**Zamysł działania programu (pominięcie błędów i błędnych danych, dla łatwiejszego zobrazowania zastąpiłem langa i langb, odpowiednio lang1 i lang2):**



Wykorzystane struktury i objaśnienie działania:

### Struktura dynamiczna słownik:

struct słownik
struct slovo* langa
struct slovo* langb
struct słownik* nextptr

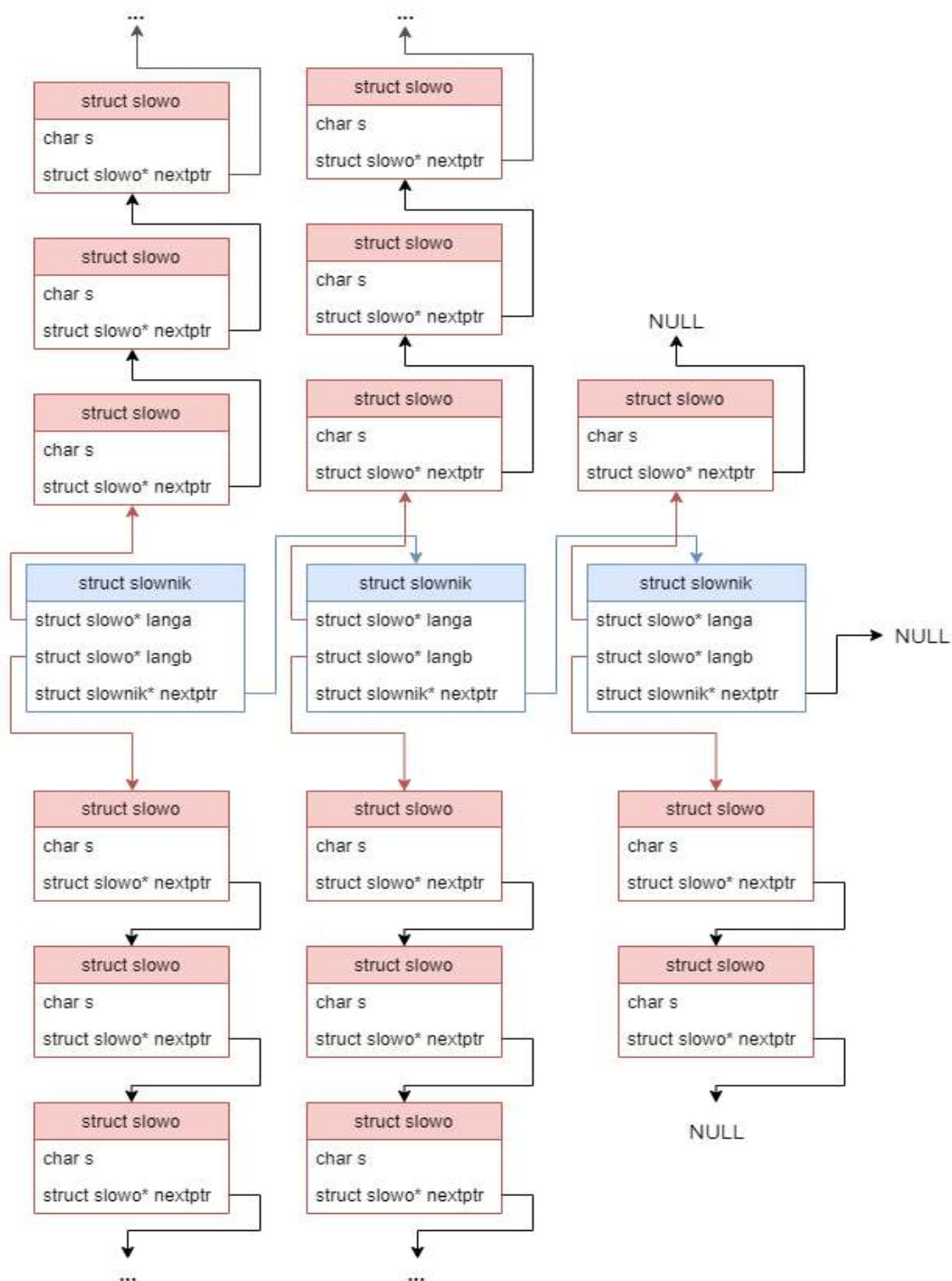
Struktura słownik ma zastosowanie jako dynamiczna lista jednokierunkowa z podwieszanymi listami jednokierunkowymi typu slovo: langa i langb, odpowiadającym: słowu z języka z którego tłumaczymy i słowu z języka na który tłumaczymy.

### Struktura dynamiczna slovo:

struct slovo
wint_t s
struct slovo* nextptr

Struktura slovo ma zastosowanie jako dynamiczna lista jednokierunkowa zawierająca tylko jeden znak (wide int do obsługi unicode) i wskaźnik na następny element listy, zdecydowałem się na wykorzystanie tego typu struktury aby zminimalizować użycie pamięci dla niepotrzebnie pustych miejsc. Miejsca takie pojawiałyby się gdyby zadeklarować tablicę char dla na przykład 50 znaków i wczytać słowo mające mniej liter.

Przykład zastosowania struktur dynamicznych w programie (dla słownika zawierającego 3 pary słów):

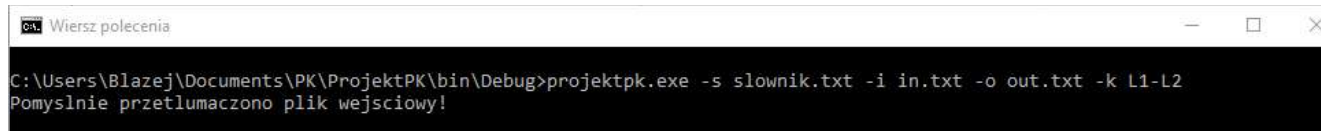


Program należy uruchomić z linii poleceń (cmd.exe) w następujący sposób:

*program.exe -s <plik z słownikiem> -i <plik z danymi wejściowymi> -o <plik w którym ma zostać zapisane tłumaczenie> -k <kierunek tłumaczenia (L1-L2 lub L2-L1)>*

Jeżeli program nie jest uruchamiany z katalogu, w którym się znajduje należy podać do niego pełną ścieżkę, tak samo należy postępować z plikiem do przetworzenia. Jednak gdy program jest w tym samym katalogu co pliki do przetworzenia nie trzeba podawać już całej ścieżki, jedynie nazwę pliku i jego rozszerzenie. W przypadku, gdy ścieżka zawiera nazwy katalogów ze spacjami należy wziąć ją w cudzysłów.

Po uruchomieniu jeśli wartości przełączników są poprawne i wszystkie przełączniki zostały wpisane program wykonuje swoje zadanie i wypisuje na ekranie „*Pomyślnie przetłumaczono plik wejściowy!*”.



```
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektprk.exe -s slownik.txt -i in.txt -o out.txt -k L1-L2
Pomyślnie przetłumaczono plik wejściowy!
```

W przypadku błędów program wyświetla następujące komunikaty:

- „*Nie udało się zaalokować pamięci programu*” - komunikat ten wyświetla się gdy programowi zabraknie pamięci, lub gdy wystąpi jakikolwiek inny problem z alokowaniem zmiennych i struktur do pamięci
- „*Podany plik nie jest plikiem tekstowym lub jest niepoprawny <nazwa pliku>*” – podany przez użytkownika plik nie jest plikiem o rozszerzeniu .txt lub jest niepoprawny na przykład: .txt lub plik.txt.
- „*Wpisana komenda, nie zawiera przełącznika -<symbol>.*” - podany przełącznik nie został wpisany przy uruchamianiu programu
- „*Wpisana komenda (kierunek tłumaczenia), bądź przełącznik są niepoprawne. Poprawny kierunek to L1-L2, lub L2-L1.*” - przełącznik k nie został wpisany przy uruchamianiu programu lub kierunek jest źle zapisany.
- „*Plik słownika zawiera niepoprawne dane.*” - plik słownika jest niepoprawny, zawiera niepoprawne znaki lub nie posiada swoich odpowiedników w drugim języku.
- „*Nie udało się otworzyć pliku słownika.*” - plik słownika nie istnieje lub program nie ma odpowiednich uprawnień aby go wczytać.
- „*Nie udało się otworzyć pliku ...*” - plik wejściowy lub wyjściowy nie istnieje albo program nie ma odpowiednich uprawnień aby go wczytać, utworzyć lub zapisać w nim (program wypisze odpowiednio który plik jest nieprawidłowy).

### Specyfikacja wewnętrzna programu

char\* Sprawdz\_poprawnosc\_komendy(char stan, int lim, char\* polecenia [])

Funkcja na podstawie podanych przez użytkownika danych ( char\* polecenia [] ) sprawdza czy w ogóle przełącznik o danym stanie ( char stan ) został wypisany i czy ścieżka do pliku jest poprawna. Jeśli przełącznik istnieje a ścieżka jest poprawna funkcja zwraca wskaźnik na ścieżkę do pliku. W przypadku błędu funkcja zakańcza działanie programu i wyświetla odpowiedni komunikat.

bool Sprawdz\_kierunek(int lim, char\* polecenia[])

Funkcja na podstawie podanych przez użytkownika danych ( char\* polecenia [] ) sprawdza czy w ogóle przełącznik o stanie k został wypisany i czy polecenie jest poprawne. Jeśli przełącznik istnieje a polecenie jest poprawne funkcja odpowiednio true lub false. W przypadku błędu funkcja zakańcza działanie programu i wyświetla odpowiedni komunikat.

slovo\* Stworz\_literke(wint\_t znak)

Funkcja tworzy nową strukturę typu slovo, na podstawie przekazanych danych przydziela jej odpowiednie wartości i zwraca wskaźnik na ową strukturę. W przypadku błędu przy alokowaniu pamięci dla struktury program jest zatrzymywany i jest wyświetlany odpowiedni komunikat.

slovo\* Zbuduj\_slovo(slovo\* head, wint\_t znak)

Funkcja inicjalizuje listę jednokierunkową złożoną z struktur typu slovo i zwraca wskaźnik na pierwszy element listy. W przypadku gdy lista jest pusta podany element staje się pierwszym jej elementem, natomiast gdy lista nie jest pusta podany element jest dodawany na jej koniec. Funkcja ta korzysta z innej funkcji – Stworz\_literke.

bool Porownaj(slovo\* pierwsze, slovo\* drugie)

Funkcja ma za zadanie porównać dwie listy typu slovo, jeśli każdy znak z obu list jest taki sam i listy są takiej samej długości zwracana jest wartość true, jeśli nie wartość false. Funkcja pomija wielkość liter.

slovo\* Usun\_slovo(slovo\* head)

Funkcja ma za zadanie usunąć całą listę jednokierunkową typu slovo i zwrócić wartość NULL. Funkcja ta spełnia to zadanie rekurencyjnie usuwając listę „od tyłu”.

slovník\* Znajdz\_slovo(slovník\* head,slovo\* word)

Funkcja ma za zadanie znaleźć i zwrócić odpowiednik słowa (word) w liście jednokierunkowej typu slovo. Wykorzystuje do tego funkcję porównującą *Porownaj*, jeśli uda się znaleźć takie samo słowo funkcja zwraca odpowiednik- langb, jeśli w całej funkcji nie ma takiego słowa funkcja zwraca wartość NULL.

struct slovník\* Stworz\_nowy\_el\_slovníka(bool rev,slovo \* a,slovo \* b)

Funkcja tworzy nową strukturę typu slovník, i przypisuje jej odpowiednio dane, to znaczy podwiesza pod tą strukturę dwie listy jednokierunkowe typu slovo. Zależnie od parametru rev są one przypisywane odpowiednio tak jak są zamieszczone w słowniku: langa=a, langb=b lub odwrotnie, wtedy tłumaczymy słowa które w słowniku są po prawej stronie na te po lewej. Sytuacją niedopuszczalną jest słowo bez odpowiednika, więc gdy funkcja napotka taką sytuację zatrzymuje pracę programu i wyświetla odpowiedni komunikat, to samo dzieje się gdy nie uda się zaalokować pamięci.

slovník \* Dodaj\_do\_listy(slovník\* head,slovník\* do\_dodania)

Funkcja inicjalizuje listę jednokierunkową złożoną z struktur typu slovník i zwraca wskaźnik na pierwszy element listy. W przypadku gdy lista jest pusta podany element staje się pierwszym jej elementem, natomiast gdy lista nie jest pusta podany element jest dodawany na jej koniec.

slovník \* Wczytaj\_slovník(char nazwa[],bool rev)

Funkcja jest jedną z głównych, która używa wyżej wymienionych funkcji, ma ona za zadanie wczytać słownik do odpowiednich struktur (slovo), stworzyć i zaalokować w pamięci listę typu slovník, stworzyć listy jednokierunkowe typu slovo i podpiąć je odpowiednio do elementów listy slovník. Całość tworzy listę list podwieszanych z czego każda pojedyncza struktura typu slovník musi mieć dwie „podlisty” typu slovo. Jeśli w pliku słownika występują niedopuszczalne znaki, funkcja przerwie działanie programu i wyświetli odpowiedni komunikat, stanie się tak również jeśli nie uda otworzyć się pliku.

void Zapisz\_do\_pliku(FILE \* foutput, slovo\* do\_przep,slovo\* element)

Funkcja ma za zadanie przepisać do pliku listę typu slovo odpowiednio *element* lub *do\_przep*. Jeśli *element* ma wartość NULL, funkcja ma za zadanie przepisać zawartość listy *do\_przep* jednak przed i po musi postawić znak nawiasu kwadratowego. W przypadku gdy *element* nie ma wartości NULL funkcja przepisuje zawartość listy *element*.

void Przetlumacz(slovník\* glowa,char we[],char wy[])

Druga z głównych funkcji, służy do odczytywania danych z pliku wejściowego (*we*), i porównywaniu tych danych z danymi w pamięci (*listy slovník*) przy pomocy funkcji *Znajdz\_slovo*, zapisuje do pliku wyjściowego (*wy*) przy pomocy funkcji *Zapisz\_do\_pliku*. Jeśli któregoś z plików nie udało się otworzyć funkcja przerywa program i wypisuje odpowiedni komunikat.

void Usun\_liste(slovník \*\* head)

Funkcja służy do usunięcia listy typu slovník i podwieszanych do niej list typu slovo. W przypadku programu jest ona wywoływana na samym końcu aby zwolnić miejsce po strukturach i zmiennych.

# Testowanie

## 1. Poprawne działanie programu dla tłumaczenia L1-L2

```
Wiersz polecenia
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i inpl.txt -o out.txt -k L1-L2
Pomyślnie przetłumaczono plik wejściowy!
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>
```

Plik pl2eng.txt

```
pl2eng.txt — Notatnik
Plik Edycja Format Widok Pomoc
ja I
ty you
on he
ona she
ono it
my we
oni they
delfin dolphin
foka seal
zebra zebra
żyrafa giraffe
mieć have
polska poland
anglia england
kamera camera
człowiek human
dziewczyna girl
chłopak boy
kobieta woman
mężczyzna man
dom house
lizak lolipop
muzyka music
programowanie programming
komputer computer
lubię like
profesjonalny professional
czipsy chips
umieć can
```

Plik wejściowy

```
inpl.txt — Notatnik
Plik Edycja Format Widok Pomoc
Ja lubię lizak, anglię, komputer, dom, foka. Ona ma dom umieć programować. My żyjemy w Polsce.
```

Plik wyjściowy (wynik)

```
out.txt — Notatnik
Plik Edycja Format Widok Pomoc
I like lolipop, england, computer, house, seal. she have house can [programować]. we [żyć] [w] poland.
```

Jak widać w wyżej załączonych zrzutach ekranu program dla poprawnych danych wykonuje swoje zadanie bez problemu.

Gdyby zmienić kierunek tekstu plik wyjściowy wyglądałby tak:

```
out.txt — Notatnik
Plik Edycja Format Widok Pomoc
[Ja] [lubię] [lizak], [anglię], [komputer], [dom], [foka]. [Ona] [ma] [dom] [umieć] [programować]. [My] [życ] [w] [Polska].
```

## 2. Niepoprawnie wpisane przełączniki i nazwy plików.

```
Wiersz polecenia
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i ineng.txt -o out.txt -k L1-L3
Wpisana komenda (kierunek tłumaczenia), błąd przełącznika. Poprawny kierunek to L1-L2, lub L2-L1.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i ineng.txt -o out.txt L1-L3
Wpisana komenda (kierunek tłumaczenia), błąd przełącznika. Poprawny kierunek to L1-L2, lub L2-L1.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i ineng.txt -o on.tx -k L1-L2
Podany plik nie jest plikiem tekstowym lub jest niepoprawny on.tx.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i ineng.txt -o on.bin -k L1-L2
Podany plik nie jest plikiem tekstowym lub jest niepoprawny on.bin.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s -i ineng.txt -o on.bin -k L1-L2
Podany plik nie jest plikiem tekstowym lub jest niepoprawny -i.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s ... -i ineng.txt -o on.bin -k L1-L2
Podany plik nie jest plikiem tekstowym lub jest niepoprawny ....

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe
Wpisana komenda, nie zawiera przełącznika -s.

C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>
```

## 3. Niepoprawny plik słownika

```
pl2eng.txt — Notatnik
Plik Edycja Format Widok Pomoc
ja I
ty you
on he
ona she
ono it
my we
oni they
delfin dolphin
Kilka spacji na początku
```

```
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpk.exe -s pl2eng.txt -i ineng.txt -o out.txt -k L1-L2
Plik słownika zawiera niepoprawne dane.
```

Dokładnie tutaj zatrzymał się program:  
Po wystąpieniu pierwszej spacji  
kolejną napotkaną zakańcza pracę programu.  
To samo stanie się gdy dodamy do słownika  
cyfrę lub znak specjalny np. %, 2 lub podamy  
trzecie słowo.

```
bool spacja = false;
while((znak = fgetc(plik)) != EOF)
{
    if ((znak < 64 && znak > 91) || (znak > 96 && znak < 123) || znak < 191)
    {
        if (spacja)
            langb = Zbuduj_slowo(langb, znak);
        else
            langa = Zbuduj_slowo(langa, znak);
    }
    else if (znak == 32 && spacja == false)
    {
        spacja = true;
    }
    else if (znak == 10 && spacja == true)
    {
        slowik = nowyStworzNowySlownik(rev, langa, langb);
        glowa = Dodaj_do_listy(glowa, nowy);
        langa = NULL;
        langb = NULL;
        spacja = false;
    }
    else
    {
        printf("Błąd słownika zawiera niepoprawne dane.\n");
        exit(0);
    }
}
```



Natomiast w przypadku gdyby zostawić słowo bez odpowiednika i pozostawić spację za nim program zatrzymuje pracę w innym miejscu i funkcji a dokładnie w funkcji Stworz\_nowy\_el\_slownika

```
pl2eng.txt - Notatnik
Plik Edycja Format Widok Pomoc
ja I
ty you
on he
ona she
ono |
```

Komunikat podany przy wywołaniu jest dokładnie taki sam jak w poprzednim przykładzie.

W tym przypadku b będzie mieć wartość NULL

```
//Funkcja tworząca nowy element typu slownik
struct slownik* Stworz_nowy_el_slownika(bool rev,slowo * a,slowo * b)
{
    struct slownik* temp= malloc(sizeof(slownik));
    if(temp!=NULL)
    {
        if(a==NULL || b==NULL)
        {
            printf("Błąd słownika zawsze niespodziewane dane.\n");
            exit(0);
        }
        if(rev)
        {
            temp->langa=b;
            temp->langb=a;
        }
    }
}
```

#### 4. Brak uprawnień lub nieistniejące pliki do odczytu

```
Nazwa
ineng.txt
inpl.txt
out.txt
pl2eng.txt
ProjektPK.exe
```

```
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpr.exe -s slownik.txt -i ineng.txt -o out.txt -k L1-L2
Nie udało się otworzyć pliku słownika.
```

Natomiast w przypadku gdy dla pliku out.txt zablokujemy możliwość zapisu (brak uprawnień):

```
C:\Users\Blazej\Documents\PK\ProjektPK\bin\Debug>projektpr.exe -s pl2eng.txt -i ineng.txt -o out.txt -k L1-L2
Nie udało się otworzyć pliku: out.txt
```

Zawartość katalogu