Programowanie i metody numeryczne

Ćwiczenia 2.

Programowanie: łańcuchy tekstowe, pliki.

Zadanie 1. `palindrome` – Palindromy.

Palindromem nazywamy wyrażenie brzmiące tak samo przy czytaniu od lewej strony do prawej, jak i odwrotnie. Przy badaniu, czy dane wyrażenie jest palindromem, nie należy brać pod uwagę wielkości liter ani znaków interpunkcyjnych. Palindromami są na przykład następujące wyrażenia: „Anna", „kajak", „O, ty z Katowic, Iwo? Tak, Zyto!"

Napisz funkcję `is_palindrome`, która przyjmuje jako argument łańcuch tekstowy i zwraca wartość `True`, jeśli jest on palindromem, lub `False` w przeciwnym przypadku.

Korzystając z tej funkcji napisz program `palindrome`, który przyjmuje jako argumenty wywołania dowolną ilość łańcuchów tekstowych i wypisuje na standardowe wyjście te z nich, które są palindromami.

Zadanie 2. `comments` – Komentarze w pliku tekstowym.

Napisz program `comments` służący do usuwania komentarzy z pliku tekstowego. Program powinien przyjmować trzy argumenty wywołania: pierwszy ma być pewnym znakiem, zaś pozostałe dwa – nazwami plików. Zadaniem programu jest przepisanie zawartości pierwszego z tych plików do drugiego z nich z pominięciem linii zaczynających się znakiem przekazanym programowi jako pierwszy argument wywołania.

Przykładowe wykonanie

Zawartość pliku wejściowego `in.txt`*:*

```
pierwsza linia
?komentarz
druga linia
```

Wywołanie:

Windows: `py comments.py ? in.txt out.txt`

macOS / Linux: `python comments.py ? in.txt out.txt`

Zawartość pliku wyjściowego `out.txt`*:*

```
pierwsza linia
druga linia
```

Zadanie 3. `bill` – Obliczanie kwoty do zapłaty.

Napisz program `bill`, którego zadaniem będzie analiza zapisanego w prostej formie rachunku i obliczenie całkowitej należności. Program ma przyjmować jako argument wywołania nazwę pliku tekstowego. Plik ten powinien zawierać w kolejnych linach nazwy zakupionych produktów, składające się z dowolnych znaków, w tym spacji, oraz ich ceny, umieszczone zawsze na końcu linii, po spacji. Ilość zakupionych produktów nie jest znana z góry. Zadaniem programu jest zsumowanie cen wszystkich produktów i wypisanie obliczonej w ten sposób całkowitej kwoty do zapłaty na standardowe wyjście.

Przykładowe wykonanie
Zawartość pliku wejściowego `grocery.txt`*:*

```
Serniczek z rodzynkami 19.90
Czekolada 90% 5.60
Pomidor 3.14
```

Wywołanie:

Windows: `py bill.py grocery.txt`
macOS / Linux: `python bill.py grocery.txt`

Wyjście:

```
28.64
```

Zadanie 4. `enigma` – Szyfrowanie z użyciem XOR.

Jedną z metod szyfrowania plików tekstowych jest zastąpienie każdego znaku bitową różnicą symetryczną (*XOR*) jego kodu ASCII z zadaną jednobajtową liczbą zwaną kluczem. Odszyfrowanie tekstu polega na ponownym obliczeniu bitowej różnicy symetrycznej zaszyfrowanych kodów z tym samym kluczem.

Napisz program `enigma` szyfrujący i deszyfrujący pliki tekstowe omówioną metodą. Program powinien przyjmować cztery argumenty wywołania. Pierwszy z nich to `-e` dla szyfrowania lub `-d` dla deszyfrowania, drugi – jednobajtowa liczba całkowita będąca kluczem, trzeci jest nazwą pliku z tekstem do zaszyfrowania/odszyfrowania, zaś czwarty – nazwą pliku, w którym ma zostać zapisany zaszyfrowany/odszyfrowany tekst. Szyfrowanie ma się odbywać znak po znaku, włączając znaki białe. Plik z zaszyfrowanym tekstem powinien zawierać oddzielone spacjami liczby – każda z nich jest bitową różnicą symetryczną kodu ASCII zaszyfrowanego znaku i klucza.

Przykładowe wykonanie 1.
Zawartość pliku wejściowego `in.txt`*:*

```
Non intratur in veritatem, nisi per caritatem.
```

Wywołanie:

Windows: `py enigma.py -e 7 in.txt out.txt`
macOS / Linux: `python enigma.py -e 7 in.txt out.txt`

Zawartość pliku wyjściowego `out.txt`*:*

```
73 104 105 39 110 105 115 117 102 115 114 117 39 110 105 39 113 98
117 110 115 102 115 98 106 43 39 105 110 116 110 39 119 98 117 39
100 102 117 110 115 102 115 98 106 41
```

Przykładowe wykonanie 2.
Zawartość pliku wejściowego `in.txt`*:*

```
73 104 105 39 110 105 115 117 102 115 114 117 39 110 105 39 113 98
117 110 115 102 115 98 106 43 39 105 110 116 110 39 119 98 117 39
100 102 117 110 115 102 115 98 106 41
```

Wywołanie:

Windows: `py enigma.py -d 7 in.txt out.txt`
macOS / Linux: `python enigma.py -d 7 in.txt out.txt`

Zawartość pliku wyjściowego `out.txt`*:*

```
Non intratur in veritatem, nisi per caritatem.
```

Zadanie 5. `caesar` – Szyfr Cezara.

Szyfr Cezara, nazywany też *szyfrem przesuwającym*, to jedna z najstarszych i zarazem najprostszych technik szyfrowania tekstu. Szyfrowanie łańcucha tekstowego szyfrem Cezara z przesunięciem $n \in \mathbb{Z}$ polega na zastąpieniu każdej z liter tego łańcucha literą występującą w alfabecie n pozycji za literą oryginalną, gdy

$n \geq 0$, lub $|n|$ pozycji przed literą oryginalną, gdy $n < 0$. Przyjmujemy przy tym, że litery wielkie i małe tworzą odrębne zbiory (a więc litera wielka zawsze zostanie zastąpiona literą wielką, zaś litera mała – literą małą) oraz że zbiory te są uporządkowane cyklicznie (tzn. przed literą A występuje litera Z, zaś za literą Z – litera A; analogicznie dla liter małych). Znaki, które nie są literami (np. cyfry, znaki specjalne), nie są w żaden sposób zmieniane. Na przykład szyfrując łańcuch tekstowy *Programowanie13w@$Pythonie* szyfrem Cezara z przesunięciem 3 dla 26-literowego alfabetu łacińskiego otrzymamy *Surjudprzdqlh13z@$Sbwkrqlh*.

Napisz funkcję `caesar`, która przyjmuje jako argumenty liczbę całkowitą oraz łańcuch tekstowy. Funkcja ta powinna szyfrować przekazany jej jako drugi argument łańcuch za pomocą szyfru Cezara z przesunięciem zadanym liczbą całkowitą przekazaną jako pierwszy argument oraz zwracać zaszyfrowany łańcuch.

Korzystając z tej funkcji napisz program `caesar`, który

- jeśli pierwszym argumentem wywołania jest `encrypt`, szyfruje przekazany mu jako trzeci argument łańcuch tekstowy za pomocą szyfru Cezara z przesunięciem zadanym liczbą całkowitą przekazaną jako drugi argument, a następnie wypisuje zaszyfrowany łańcuch na standardowe wyjście,
- jeśli pierwszym argumentem wywołania jest `decrypt`, odszyfrowuje przekazany mu jako trzeci argument łańcuch tekstowy, zaszyfrowany za pomocą szyfru Cezara z przesunięciem zadanym liczbą całkowitą przekazaną jako drugi argument, a następnie wypisuje odszyfrowany łańcuch na standardowe wyjście.

Przyjmij, że litery pojawiające się w łańcuchach tekstowych należą do 26-literowego alfabetu łacińskiego.

Zadanie 6. `brackets` – Balansowanie ciągów nawiasów.

Ciąg znaków złożony z nawiasów `(`, `)`, `[`, `]` i `{`, `}` uważamy za *zbalansowany*, jeśli każdemu z nawiasów otwierających odpowiada właściwy nawias zamykający oraz jeśli nawiasy są poprawnie zagnieżdżone. Na przykład ciągi: `()[{}]`, `(){[]{([])}}` są zbalansowane, zaś ciągi `)(`, `[{]}`, `({[})` – nie.

Napisz program `brackets`, który:

- jeśli pierwszym argumentem wywołania jest `check` – sprawdza, czy ciąg nawiasów przekazany mu jako drugi argument wywołania jest zbalansowany i wypisuje na standardowe wyjście stosowną informację,
- jeśli pierwszym argumentem wywołania jest `fix` – wypisuje na standardowe wyjście liczbę nawiasów, które trzeba dopisać w ciągu złożonym wyłącznie z nawiasów okrągłych `(` i `)`, przekazanym jako drugi argument wywołania, by ciąg ten był zbalansowany,
- jeśli pierwszym argumentem wywołania jest `list` – wypisuje na standardowe wyjście wszystkie zbalansowane ciągi nawiasów o długości $2n$, gdzie n jest liczbą całkowitą przekazaną jako drugi argument wywołania, złożone wyłącznie z nawiasów okrągłych `(` i `)`.

Opracowanie: Bartłomiej Zglinicki.