Propozycje zadań do realizacji

Uwaga: Wszystkie zadania należy wykonać w języku Python i C++.

- ### Zadanie 1 – Konwersja

  Przedmiotem zadania jest implementacja metod konwersji liczb z postaci dziesiętnej na dwójkową i odwrotnie. W tym celu należy napisać dwa programy:

  `dec2bin.cpp`
  `bin2dec.cpp`

  Program `dec2bin.exe` powinien przyjmować jako argument liczbę w systemie dziesiętnym i wypisywać na standardowym wyjściu odpowiadającą jej liczbę w systemie binarnym.
  Program `bin2dec.exe` powinien przyjmować jako argument liczbę w systemie dwójkowym i wypisywać na standardowym wyjściu odpowiadającą jej liczbę w systemie dziesiętnym, przykładowo:

```
dec2bin.exe 7
111
dec2bin.exe 8
1000
dec2bin.exe 50
110010
bin2dec.exe 110011
51
bin2dec 101
5
bin2dec 11111111
255
```

- ## Zadanie 2 – Kalkulator bitowy

  Celem tego zadania jest demonstracja działania podstawowych, dwuargumentowych operatorów bitowych (`|`, `&`, `^`, `<<`, `>>`). Należy zaimplementować program `bitCalc` przyjmujący jeden parametr: napis (ujęty w znaki cudzysłowu) zawierający trzy elementy (oddzielane spacjami): pierwszy argument, operator i drugi argument, przykładowo:

  ```
  bitCalc.exe "7 << 1"
  14
  bitCalc.exe "69 & 15"
  5
  ```

  Oba argumenty powinny być traktowane jak zmienne liczbowe (w zapisie dziesiętnym) a wynik operacji bitowej należy wypisać na standardowe wyjście.

- ## Zadanie 3 – Litery bitowo parzyste

  Przesyłanie danych często wiąże się z ryzykiem przekłamań i błędów transmisji. Prostą metodą pozwalającą na ich wykrycie jest stosowanie tzw. bitu parzystości, czyli dodatkowego bitu, którego wartość jest ustawiana przez nadajnik tak, aby liczba wszystkich bitów równych jeden była zawsze parzysta. Jeśli odbiornik stwierdzi, że liczba jedynek w odebranej porcji danych jest nieparzysta – oznacza to, że podczas transmisji doszło do przekłamania.

  W tym zadaniu zaimplementujemy pewien prosty wariant tej metody pozwalający na zabezpieczanie przed przekłamaniami ciągów złożonych z samych liter (małych i wielkich). Ponieważ każda mała litera różni się od odpowiadającej jej wielkiej tylko jednym bitem (piąty bit – o wadze 32), więc zmieniając w razie potrzeby małą literę na wielką lub odwrotnie, możemy zawsze zagwarantować, że uzyskamy literę "bitowo parzystą", tj. że binarna reprezentacja kodu ASCII tej litery będzie mieć parzystą liczbę bitów.

  Wejście:

  W pierwszym wierszu standardowego wejścia znajduje się jedna dodatnia liczba całkowita N. Dalej następuje N wierszy, z których każdy zawiera ciąg małych i wielkich liter o dowolnej długości (długość każdego wiersza może być inna).

  Wyjście:

  Na standardowe wyjście należy wypisać N wierszy o takiej samej długości jak odpowiadające im wiersze wejściowe. Zawartość każdego n-tego wiersza wyjściowego powinna odpowiadać zawartości n-tego wiersza wejściowego, z dokładnością do

wielkości liter, tzn. dozwolone są tylko zamiany małej litery na odpowiadającą jej wielką literę bądź odwrotnie. Każda taka zamiana powinna być wykonana tylko w przypadku, gdy liczba bitów w reprezentacji binarnej kodu ASCII wejściowej litery jest nieparzysta, tak aby zagwarantować, że wszystkie litery wypisywane na wyjście będą "bitowo parzyste".

Przykład:
Dla danych wejściowych:

```
3
Ala
ma
kota
```

poprawnym wynikiem jest:

```
AlA
MA
KotA
```

Wyjaśnienie:
Kod małej litery "a" to 97 (binarnie: 01100001), a wielkiej litery "A" to 65 (binarnie: 01000001), zatem liczba jedynek w literze małej jest nieparzysta (3), a w wielkiej – parzysta (2). W związku z tym wszystkie wystąpienia małej litery "a" zostały zastąpione wielką literą.

Kody ASCII liter występujących w napisach wyjściowych są następujące:

litera	kod dziesiętnie	kod binarnie	ilość jedynek
A	65	01000001	2
l	108	01101100	4
M	77	01001101	4
K	75	01001011	4
o	111	01101111	6
t	116	01110100	4

a zatem liczba jedynek w każdym przypadku spełnia warunki zadania.