Programowanie proceduralne — lista zadań laboratoryjnych II

1. Czytamy wpisane z konsoli znaki, sprawdzając czy są to cyfry 0, 1, ..., 9. Pierwszy znak różny od cyfry (np. spacja lub koniec wiersza) kończy proces czytania. Wczytany ciąg należy zamienić "ręcznie" na liczbę typu `int` i kontrolnie wydrukować.

2. Podobnie jak wyżej, lecz teraz liczba może być poprzedzona znakiem `+` lub `-`.

3. Tym razem należy utworzyć liczbę typu `float` z wczytanego ciągu znaków, np. `-2387.955`.

4. Powtórzmy zadanie 2, dobudowując do niego zabezpieczenie przed nadmiarem: należy zadbać, aby wczytana liczba nie przekroczyła dopuszczalnego zakresu typu `int`, $-2^{31} \leqslant n \leqslant 2^{31} - 1$, czyli $-2147483648 \leqslant n \leqslant 2147483647$.

5. Zadanie polega na rozwiązaniu odwrotnego problemu: mając liczbę typu `int` (np. wczytaną z konsoli formatem `"%i"`) należy zamienić ją "ręcznie" na ciąg znaków i tak wydrukować. Uwzględnić przy tym ew. znak `-`.

6. Czytamy liczbę $1 \leqslant n \leqslant 3999$ i zamieniamy ją na postać rzymską, np. 1957 — MCMLVII.

7. Zadanie odwrotne: wczytaną liczbę w postaci rzymskiej zamieniamy na `int` i drukujemy.

8. Czytamy kolejne znaki tekstu wpisane z konsoli i zapisujemy je w pliku tekstowym. Jako sygnał zakończenia danych przyjmujemy wiersz rozpoczynający się znakiem `*`.

9. Należy odczytać zawartość pliku tekstowego znak po znaku i wyświetlić go na konsoli.

10. Program czyta plik tekstowy znak po znaku, obliczając liczbę wszystkich liter `'a'...'z', 'A'...'Z'` oraz liczbę wystąpień poszczególnych liter. Opuszczamy przy tym znaki przestankowe, odstępy i znaki zmiany wiersza. Wynikiem działania programu jest statystyka częstości wystąpień poszczególnych liter, np. `a`: 7.54%, `b`: 2.84%, itd.

11. Napisać program szyfrujący podany w pliku tekst za pomocą szyfru przesunięciowego Cezara z ustalonym przesunięciem `n`. Dla uproszczenia zakładamy, że tekst składa się z dużych liter `'A'...'Z'` bez polskich znaków diakrytycznych. Spacje i znaki przestankowe nie podlegają szyfrowaniu.

12. Wczytać łańcuch i sprawdzić czy jest on palindromem (tj. słowem, które czytane w obu kierunkach jest identyczne, np. `ala, kajak, zaraz`). Palindromem może być także dłuższy napis z pominięciem spacji, np. `"Kobyła ma mały bok"`.

13. Program wczytuje liczbę `n`, a następnie kolejno `n` par łańcuchów "nazwisko imię", umieszczając je w tablicy. Należy posortować wczytane dane w porządku alfabetycznym wg nazwisk i wyświetlić je. W pliku `dane2.txt` znajdują się przykładowe dane (100 nazwisk i imion).

14. Do poprzedniego zadania dopisać funkcję, która sprawdza czy osoba o podanym z konsoli nazwisku znajduje się na utworzonej wcześniej liście. Zastosować algorytm wyszukiwania połówkowego.

15. Jako dane czytamy 2 łańcuchy `x` i `y`. Należy wyznaczyć najdłuższe wspólne podsłowo w nich zawarte. Np.

`x` = abbab<u>bbaaba</u>aaaa i `y` = bbabaaa<u>bbaaba</u>ba.

Jako wynik, algorytm powinien podać pozycję `p` początkowej litery znalezionego podsłowa w łańcuchu `x` i jego długość `n`. Dla przykładowych danych powyżej są to liczby `p=6` i `n=7`.

Przydatne funkcje

— Czytanie i pisanie znaków (plik nagłówkowy stdio.h): `getchar(), putchar(c), scanf("%c" ...), printf("%c" ...)`

— Funkcje i operacje na znakach (plik nagłówkowy ctype.h): `isalnum(c), isalpha, islower, isupper, isdigit, ispunct, isgraph, ... tolower, toupper`

— Operacje na plikach (plik nagłówkowy stdio.h): `fopen(...), fscanf(...), fprintf(...), fclose(...)`

— Czytanie i pisanie łańcuchów (plik nagłowkowy stdio.h): `gets(...), puts(...), fgets(...), fputs(...), scanf("%s" ...), printf("%s" ...)`

— Operacje na łańcuchach (plik nagłowkowy string.h): `strlen(...), strcmp(...), strcat(...), strcpy(...)`