Programowanie i metody numeryczne

Zadania – seria 5.

Łańcuchy tekstowe.

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: „Anna", „kajak", „O, ty z Katowic, Iwo? Tak, Zyto!"

Napisz funkcję

```
bool isPalindrome(const std::string& str)
```

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 te z nich, które są palindromami. Jeśli wśród argumentów wywołania będzie przełącznik `/a`, `/all`, `-a` lub `--all`, program powinien wypisać wszystkie przekazane mu łańcuchy, informując, które z nich są palindromami, a które nie.

Zadanie 2. `simplecalc` – Prosty kalkulator.

Napisz program `simplecalc`, który przyjmuje jako pierwszy argument wywołania przełącznik, zaś jako kolejne argumenty wywołania – dowolnie wiele liczb zmiennoprzecinkowych. Program powinien:

- jeśli przełącznikiem jest `/a`, `/add`, `-a` lub `--add`, wypisać sumę liczb stanowiących pozostałe argumenty wywołania,
- jeśli przełącznikiem jest `/m`, `/mul`, `-m` lub `--mul`, wypisać iloczyn liczb stanowiących pozostałe argumenty wywołania,
- jeśli przełącznikiem jest `/?`, `-h` lub `--help`, wypisać informację o poprawnej składni wywołania programu.

Gdy wśród argumentów wywołania programu, na dowolnym miejscu, znajdzie się dodatkowo przełącznik `/e`, `/expression`, `-e` lub `--expression`, program powinien oprócz wyniku odpowiedniego działania wypisać również obliczane wyrażenie (np. `1 * 2 * 3 = 6`). Jeśli użytkownik wywoła program z niepoprawnymi argumentami, powinien on wyświetlić informacje o poprawnej składni swego wywołania (jak przy pojawieniu się przełącznika `/?`).

Zadanie 3. `todec` – Konwersja liczb na system dziesiętny.

Napisz funkcję

```
int toDec(int n, const std::string& str)
```

której zadaniem jest konwersja liczby zapisanej w danym liczbowym systemie pozycyjnym na liczbę w systemie dziesiętnym. Funkcja ta przyjmuje dwa argumenty: `n` jest liczbą całkowitą oznaczającą podstawę systemu liczbowego, w którym zapisana jest konwertowana liczba, zaś `str` – referencją do łańcucha tekstowego zawierającego zapis konwertowanej liczby w tym systemie. Wartością zwracaną przez tę funkcję powinna być wartość konwertowanej liczby zapisana w systemie dziesiętnym.

Korzystając z tej funkcji, napisz program `todec`, który przyjmuje jako argumenty wywołania liczbę całkowitą oznaczającą podstawę systemu liczbowego oraz łańcuch tekstowy zawierający liczbę zapisaną w tym systemie oraz wypisuje na standardowe wyjście wynik konwersji na system dziesiętny.

Zadanie 4. `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 jest zbalansowany i wypisuje na ekranie stosowną informację,
- jeśli pierwszym argumentem wywołania jest `fix`, wypisuje liczbę nawiasów, które trzeba dopisać w ciągu złożonym wyłącznie z nawiasów okrągłych `(` i `)`, przekazanym jako drugi argument, by ciąg ten był zbalansowany,
- jeśli pierwszym argumentem wywołania jest `list`, wypisuje wszystkie zbalansowane ciągi nawiasów złożone wyłącznie z nawiasów okrągłych `(` i `)` o długości $2n$, gdzie n jest liczbą całkowitą przekazaną jako drugi argument wywołania.

Zadanie 5. `caesar` – Szyfr Cezara. *

Szyfr Cezara, nazywany też *szyfrem przesuwającym*, to jedna z najstarszych i zarazem najprostszych technik szyfrowania tekstu. Kodowanie łańcucha tekstowego szyfrem Cezara z przesunięciem n (dodatnim lub ujemnym) polega na zastąpieniu każdej z liter literą występującą w alfabecie o n pozycji dalej. Przyjmujemy przy tym, że litery wielkie i małe są kodowane niezależnie oraz że przed literą A następuje litera Z, zaś za literą Z – litera A (analogicznie dla liter małych). Znaki, które nie są literami, powinny zostać pominięte. Deszyfrowanie wiadomości zaszyfrowanej szyfrem Cezara z przesunięciem n jest równoważne szyfrowaniu tej wiadomości z przesunięciem $-n$.

Napisz funkcję

```
std::string caesar(int n, const std::string& str)
```

która przyjmuje jako argumenty przesunięcie n oraz łańcuch tekstowy do zaszyfrowania i zwraca zaszyfrowany łańcuch.

Korzystając z tej funkcji, napisz program `caesar`, który przyjmuje jako argumenty wywołania przesunięcie n oraz łańcuch tekstowy do zaszyfrowania i wypisuje na standardowe wyjście zaszyfrowany łańcuch.

Opracowanie: Bartłomiej Zglinicki.