Laboratorium 3

1. Funkcja exponent

   Niech $s_i = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^i}{i!}$. Wiedząc, że $\lim_{i\to\infty} s_i = \exp(x)$, napisz obiektowy program, który oblicza przybliżoną wartość $\exp(x)$ dla zadanej wartości argumentu x. Obliczenia powinny się zakończyć w sytuacji, gdy $|s_i - s_{i-1}| < \varepsilon$. Wartości x oraz ε są zadane na standardowym wejściu. Przemyśl charakter i dostępność poszczególnych składowych klasy.

   Przykład

   Dane wejściowe
   ```
   2    0.01   // x i epsilon
   ```

   Dane wyjściowe
   ```
   7.387301
   ```

2. Przetwarzanie tekstów

   Na standardowym wejściu, w dwóch kolejnych wierszach zapisano:
   – tzw. pomocniczy łańcuch znaków,
   – ścieżkę dostępu do pliku tekstowego, w którym są przechowywane tzw. główne łańcuchy znaków (każdy w odrębnym wierszu, nie więcej niż 50).
   Sprawdź, ile razy pomocniczy łańcuch znaków mieści się w każdym z łańcuchów głównych (dwa różne wystąpienia łańcucha pomocniczego nie mogą się pokrywać na żadnym ze znaków).
   Zadanie wykonaj przy użyciu typu klasowego `Teksty` zawierającego, między innymi:
   – pole prywatne `pomoc`, przechowujące pomocniczy łańcuch znaków,
   – metodę prywatną `ile_razy`, sprawdzającą liczbę różnych wystąpień łańcucha pomocniczego w łańcuchu głównym,
   – pole publiczne `zestawienie`, przechowujące zbiorczą informację o liczbie wystąpień łańcucha pomocniczego we wszystkich łańcuchach głównych,
   – pole publiczne `najlepszy`, przechowujące łańcuch główny o największej wykrytej liczbie wystąpień łańcucha pomocniczego,
   – klasyczny konstruktor 2-argumentowy, z argumentami: pomocniczym łańcuchem znaków i ścieżką dostępu do pliku wejściowego,
   – konstruktory kopiujący i przenoszący,
   – destruktor,
   – metodę publiczną `czytaj`, odpowiedzialną za wczytanie wszystkich umieszczonych w pliku wejściowym łańcuchów głównych i zrealizowanie żądanych obliczeń.
   Wyprowadź na wyjście informację o liczbie wystąpień ciągu pomocniczego w poszczególnych ciągach głównych, numerowanych kolejno od `1`, oraz najlepszy ciąg główny.
   Zilustruj tworzenie obiektów typu klasowego: `Teksty` przy użyciu zadanego konstruktora klasycznego, `Teksty_l` przy użyciu zadanego konstruktora kopiującego, `Teksty_r` przy użyciu zadanego konstruktora przenoszącego.

Przykład

Dane wejściowe
```
abca
dane.txt
```

Zawartość pliku dane.txt
```
abcabca
nsabcatdftabca
abccabc
abcaabcaewrxrabcayyabc
```

Dane wyjściowe
```
1 – 1
2 – 2
3 – 0
4 – 3
abcaabcaewrxrabcayyabc
```

3. PW/MC lab/zad1 (bez podpunktu d)

Zadanie obowiązkowe

4. Pomiary czasu

   Zaprojektuj klasę `PomiarCzasu`, która pozwala na pomiar czasu wykonywania algorytmów. Klasa powinna zawierać metody:

   - `Start()` i `Stop()`, które określają moment, odpowiednio początkowy i końcowy wykonywania algorytmu,
   - `PodajCzasPocz()` i `PodajCzasKon()`, które wyświetlają wyniki obu pomiarów na konsoli,
   - `IleCzasu()`, obliczającą czas wykonywania algorytmu.

   Do odmierzania czasu zastosuj funkcję `GetTickCount()` z biblioteki `windows.h`. Funkcja ta zwraca liczbę milisekund, które upłynęły od startu systemu. Przemyśl liczbę i postać pól w klasie.
   Przeanalizuj, jakie korzyści daje zastosowanie klasy zamiast osobnych funkcji. Zastanów się, które składowe powinny być publiczne, a które prywatne.
   Wykorzystaj utworzoną klasę do pomiaru średnich czasów wykonywania trzech algorytmów sortowania: bąbelkowego, stogowego i szybkiego `300`-elementowego ciągu liczb (patrz zadanie 2.7).

Zadanie dla chętnych

5. Symulator budowy samochodu

   Zaprojektuj tę część symulatora budowy samochodu osobowego, która jest odpowiedzialna za wstawienie kół do samochodu. Posłuż się klasami `felga`, `opona` oraz `kolo`. Klasa `kolo` winna zawierać m.in. dwa pola obiektowe o typach `opona` i `felga`.
   W definicji wymienionych klas należy uwzględnić następujące istotne cechy opony, felgi i koła samochodowego:

Felga:
– `waga`,
– `szerokosc_felgi` (w milimetrach),
– `zewnetrzna_srednica_felgi` (w milimetrach).

Opona:
– `waga`,
– `szerokosc_opony` (w milimetrach),
– `srednica_osadzenia_na_feldze` (podawana w calach średnica felgi, na jaką może być zamontowana opona).

Koło:
– `waga_kola` (sumaryczna waga koła, wyznaczana na podstawie wagi opony oraz wagi felgi).

Uwaga!
Przy montażu koła, do felgi dobierana jest odpowiednia opona. Zatem, obiekty typu `kolo` powinny implementować takie funkcje, jak: `zamontuj_felge` oraz `zamontuj_opone` (funkcje te przyjmują odpowiednie obiekty w roli parametrów wejściowych). W klasach `felga` i `opona` potrzebne są funkcje udostepniające (do wglądu) parametry felgi i opony.
Przy dodawaniu do obiektu `kolo` obiektu `opona` należy sprawdzić, czy szerokość felgi oraz jej średnica pozwalają założyć tę oponę na felgę (dopuszczalna różnica rozmiarów to ±`10` mm). Uwzględnij próbę założenia nieodpowiedniej opony, informując o tym użytkownika komunikatem tekstowym.
Ponieważ model koła będzie wykorzystywany do symulacji poruszania się auta, obiekt typu `kolo` powinien również zawierać pola `calkowita_masa`, a także `tarcie_toczne`. Wartość tego ostatniego wyznacza się ze wzoru:

$$T = f \cdot N/R$$

gdzie:
– f oznacza współczynnik tarcia opony o podłoże,
– N – siłę nacisku na podłoże masy pomniejszonej samochodu, uzyskiwanej z jego masy całkowitej przez odliczenie masy kół,
– R – promień koła samochodu.