09. Biblioteka STL; organizacja kodu

1 Biblioteka STL

Dokumentacja biblioteki standardowej znajduje się pod adresem: `http://cplusplus.com/reference/`.

Do najpopularniejszych bibliotek należą:

- `<vector>` — wektor,
- `<list>` — lista dwukierunkowa,
- `<map>` — tablica asocjacyjna,
- `<set>` — zbiór,
- `<algorithm>` — podstawowe algorytmy związane z kontenerami,
- `<string>` — łańcuchy znaków,
- `<iostream>` — obsługa we/wy,
- `<fstream>` — narzędzia z we/wy plikowym,
- `<exception>` — obsługa wyjątków.

Zadanie 01 Jaka jest różnica między kontenerami: `vector`, `list`, `set`, `map`? Które w jakich sytuacjach powinniśmy wykorzystywać?

Zapoznaj się z kodem źródłowym znajdującym się w pliku `pob09-stem.cpp`. Podczas uruchomienia programu, za pomocą klawiczy `CTRL + d` można zakończyć wczytywanie danych ze standardowego wejścia lub można wykorzystać potoki:

```
1 > g++ pob09-stem.cpp -o stem
2 > echo "Ala ma kota" | ./stem
```

Zadanie 02 Zmodyfikuj funkcję `split(std::string, char)` tak, aby zwracała wektor wyrazów (`std::vector`) zamiast pisać je bezpośrednio na standardowe wyjście. Użyj jej w funkcji `main` programu.

Zadanie 03 Zdefiniuj funkcję `join`, która będzie działała odwrotnie do funkcji `split`, tj. połączy wyrazy z wektora przekazanego jako pierwszy argument znakiem przekazanym jako drugi argument i zwróci je w postaci łańcucha znaków `std::string`.

Przetestuj jej działanie w funkcji `main` programu.

Zadanie 04 Stwórz klasę `Stemmer`, dzięki której będzie możliwe sprowadzanie odmienionych wyrazów do ich formy podstawowej na podstawie słownika zawierającego pary wyrazów: *odmieniony wyraz — forma podstawowa wyrazu*.

Słownik ten będzie wczytywany z pliku, w którym pary wyrazów są oddzielone znakiem tabulacji (patrz plik `pl.stupid.dict`).

Klasa powinna posiadać:

- atrybut prywatny reprezentujący słownik w postaci tablicy asocjacyjnej (`std::map`),
- jednoargumentowy konstruktor przyjmujący nazwę pliku ze słownikiem (do wczytywania pliku wykorzystaj obiekt `std::ifstream`),
- publiczną metodę `stem(std::string word)`, która zwróci formę podstawową dla wyrazu `word`, jeżeli występuje on w słowniku, a w przeciwnym wypadku zwróci ten sam wyraz.

Napisz fragment kodu testujący klasę `Stemmer` oraz metodę `stem`.

Zadanie 05 Rozszerz klasę `Stemmer` o metodę `stem_text`, która zamiast na pojedynczym wyrazie będzie działała na tekście (zakładamy, że wszystkie wyrazy oddzielone są znakiem spacji). W jej implementacji wykorzystaj funkcje `split`, `join` oraz metodę `stem`.

Zmodyfikuj funkcję `main` tak, aby program wypisywał formy podstawowe wyrazów z tekstu pobranego ze standardowego wejścia.

Zadanie 06 Co robi funkcja `parse_options(int, const char**)`? Wykorzystaj ją do wczytania nazwy pliku ze słownikiem jako argument programu.

Przykładowa kompilacja i uruchomienie:

```
> g++ pob09-stem.cpp -o stem
> echo "Ala ma kota i psa" | ./stem pl.stupid.dict
```

2 Podział kodu na pliki

W języku C++ kod zapisany jest w plikach dwóch rodzajów:

- plikach podstawowych o rozszerzeniu `cpp`, `c` itd.,
- plikach nagłówkowych o rozszerzeniu `hpp`, `h` itd.

Zazwyczaj każda klasa posiada swój plik nagłówkowy `hpp` z jej deklaracją:

```
class Foo {
public:
    Foo(int x);
    float bar();

private:
    char baz;
};
```

oraz plik podstawowy cpp z definicjami składowych:

```
#include "foo.hpp"

Foo::Foo(int x) { /* do something... */ }
float Foo::bar() { /* do something... */ }
```

Uwaga: chcąc wykorzystać klasę w innym pliku źródłowym należy dołączyć jej plik nagłówkowy (np. `#include "foo.hpp"`), nie podstawowy.

Zadanie 07 Podziel kod programu na co najmniej cztery pliki:

- `stemmer.hpp` — z deklaracjami klasy `Stemmer` i jej składowych,
- `stemmer.cpp` — z definicjami konstruktora i metod klasy,
- `text.hpp` — z funkcjami pomocniczymi związanymi z przetwarzaniem tekstu: `split`, `join` oraz `lower`; funkcje powinny zostać zdefiniowane w przestrzeni nazw (`namespace`) o nazwie `text`,
- `main.cpp` — z funkcją `main` programu i funkcjami pomocniczymi.

Przykładowa kompilacja programu z wieloma plikami:

```
> g++ stemmer.cpp main.cpp -o stem
```

2.1 Instrukcje preprocesora

Instrukcja `#include` dołącza plik o podanej nazwie. Aby uniknąć problemu wielokrotnych deklaracji tych samych klas lub funkcji podczas dołączania pliku nagłówkowego do wielu plików źródłowych programu, w pliku nagłówkowych definiuje się strażnika (ang. *include guard, header guard*), wykorzystując makrodefinicje `#ifndef` oraz `#define`:

```
#ifndef __FOO_HPP__  // jezeli stala nie jest zdefiniowana...
#define __FOO_HPP__  // unikatowa stala zdefiniowana przy pierwszym
                     // dolaczeniu pliku
class Foo {
public:
    Foo(int x) { //...
};

#endif               // koniec instrukcji warunkowej
```

Zadanie 08 W pliku z główną funkcją programu tekst wczytywany ze standardowego wejścia powinien być najpierw sprowadzony do małych liter. Wykorzystaj funkcję `lower` poprzez dołączenie odpowiedniego pliku. Plik `text.hpp` powinien być dołączany zarówno w `stemmer.cpp`,

jak i `main.cpp`. Czy jest możliwe skompilowanie tak zorganizowanego programu? Dlaczego tak się dzieje?

Użyj odpowiednich makrodefinicji, aby naprawić problem. Uwaga: konieczne może okazać się rozdzielenie kodu z pliku `text.hpp` na pliki nagłówkowy i podstawowy.