Lista 7 – funktory (nie było na wykładzie, ale *c'est la vie*)

1. Aby obliczyć, ile razy jakaś liczba występuje w kontenerze STL, np. wektorze, można użyć obiektów funkcyjnych lub wyrażeń lambda:

```
#include <iostream>
#include <vector>
#include <algorithm>    // count_if, sort

struct
{
  bool operator()(int k) const { return k == 5; }
};

int main()
{
  std::vector<int> v = {1, 5, 16, 12, 35, 6, 13, 45, 5, 8};
  std::cout << std::count_if(begin(v), end(v), Zliczaj_5()) << "\n";
  std::cout << std::count_if(begin(v), end(v), [](int k) { return k == 5; }
            << "\n";
}
```

   a) Zaprojektuj strukturę **`Sortuj`**, w której będzie zdefiniowany **dwuargumentowy** operator wywołania funkcji, i użyj go do uporządkowania wektora rosnąco wg wartości cyfry jedności w zapisie dziesiętnym liczby. Czyli np. powyższy ciąg może być uporządkowany jako `1 12 13 5 35 45 5 16 6 8` lub `1 12 13 35 45 5 5 6 16 8`. Liczy się tylko wartość cyfry jedności.
   Opis `std::sort` znajduje się np. tu: https://en.cppreference.com/w/cpp/algorithm/sort – por. przykładowy program na końcu.

   b) Jak już się uporasz z punktem a), to rozwiąż ten samego problem za pomocą wyrażenia lambda.
   Wskazówka 1: Cóż z tego, że tego nie było na wykładzie. To jest zadanie na spostrzegawczość i eksperymentowanie. Zwróć uwagę na **identyczne fragmenty** w rozwiązaniu za pomocą obiektu funkcyjnego i wyrażenia lambda w podanym przeze mnie przykładzie.
   Wskazówka 2: rozwiązanie jest łatwe, zwłaszcza gdy już się je zna.
   Wskazówka 3. `operator ()` odruchowo definiuj z atrybutem `const`. Czasami nie jest to potrzebne, może być niepożądane, ale jeśli jest konieczne, a ty o tym `const` zapomnisz, to zwykle kompilator wyświetli mnóstwo komunikatów, które będą bardzo trudne w interpretacji.

Pytania kontrolne

1. Co to jest funktor? Co to jest obiekt funkcyjny? https://en.wikipedia.org/wiki/Function_object
2. W moim przykładzie pojawia się wyrażenie `Zliczaj_5()`. Co ono reprezentuje? Jaki jest czas życia tak definiowanego obiektu (tzn. w którym momencie wywołałby się jego destruktor, gdyby był zdefiniowany)?
3. Przy przejściu z języka w wersji A do nowszej wersji B kompilatory często, choć nie zawsze, tłumaczą program z wersji B do wersji A i dopiero potem go kompilują. Na przykład pierwsze kompilatory C++ tłumaczyły programy w C++ na język C i dopiero wtedy je kompilowały. Czy Twoim zdaniem napisany w ten sposób kompilator C++ zamieniałby wyrażenia lambda, które pojawiły się w języku C++11, na „anonimowe funkcje” czy raczej na (anonimowe) obiekty funkcyjne?
   Wskazówka: Jeśli nie nasz odpowiedzi, przeczytaj Uwagi i wtedy tu wróć.

Uwagi

1. Zapamiętaj ścisły związek lambd z obiektami funkcyjnymi. Dziś już mało kto używa obiektów funkcyjnych bezpośrednio – lambdy są wygodniejsze w użyciu, ale trudniejsze w zrozumieniu sposobu działania, jeśli z tyłu głowy nie ma się sposobu, w jaki są tłumaczone na obiekty funkcyjne. Bardzo dużo algorytmów biblioteki standardowej C++ zaprojektowano do współpracy z obiektami funkcyjnymi / lambdami / wskaźnikami na funkcje – kiedyś królowały obiekty funkcyjne, dziś – wyrażenia lambda.
2. Jeśli zastanawiasz się, do czego służą puste nawiasy kwadratowe [] w definicji lambdy w powyższym przykładzie, to zwróć uwagę na to, że zbiór składowych z danymi w strukturze **`Zliczaj_5`** również jest pusty. W tych nawiasach można zdefiniować, jakie składowe (z danymi) ma mieć obiekt funkcyjny, na który zostanie przetłumaczone wyrażenie lambda, i które z tych składowych mają tam być umieszczone jako składowe „zwykłe”, a które jako referencje. Jedyną funkcją składową takiego funktora jest operator funkcyjny, który w lambdzie definiuje się w sekcjach zawartych w nawiasach okrągłych () i klamrach {}. Typ wartości tego operatora to zazwyczaj `auto`, dlatego (zazwyczaj) nie widać go w definicji lambdy.