30 października 2021

Assignment 2

W poniższym kodzie szablon funkcji **part** deklaruje, jako trzeci parametr f typu **FUN**, *cokolwiek* co da się wywołać (wskaźnik funkcyjny, lambda) z argumentem typu **T** i zwraca **bool** (takie funkcje nazywamy *predykatami*).

Uzupełnij kod programu, tak, aby dał się skompilować i wykonać.

Funkcja (szablon) **printTab** ma za zadanie wydrukować w ładnej formie przekazaną tablicę (elementy w jednym wierszu, oddzielone spacjami).

Funkcja (szablon) **part** ma za zadanie tak poprzestawiać elementy przekazanej tablicy arr, aby wszystkie elementy, dla których predykat f jest spełniony (tzn. f wywołany z wartością tego elementu jako argumentem zwraca **true**) znalazły się na lewo od wszystkich elementów, dla których ten predykat nie jest spełniony. Jako argumentu odpowiadającego parametrowi f typu **FUN** można użyć wskaźnika do funkcji typu **T→bool** (jak w linii 20) lub lambdy o takiej sygnaturze. W wynikowej tablicy względna kolejność elementów w ramach tych, które predykat spełniają i tych, które go nie spełniają, jest dowolna. Funkcja zwraca indeks pierwszego elementu, który *nie* spełnia predykatu; zauważ, że jest to jednocześnie liczba elementów, które predykat *spełniają* (być może 0 lub size).

download *FunTmpl.cpp*

```
1  #include <iostream>
2  #include <string>    // for tests
3  #include <ctime>     // time
4  #include <utility>   // std::swap may be useful
5
6  template <typename T, typename FUN>
7  size_t part(T* arr, size_t size, FUN f) {
8      // ...
9  }
10
11 template <typename T>
12 void printTab(const T* t, size_t size) {
13     // ...
14 }
15
16 bool isEven(int e) { return e%2 == 0; }
17
18 int main() {
19     size_t ind = 0;
20
21     int a1[] = {1,2,3,4,5,6};
22     ind = part(a1,6,isEven);
23     std::cout << "ind = " << ind << " ";
```

```
24      printTab(a1,6);
25
26      int a2[] = {1,2,3,4,5,6};
27        // lambda as argument: a predicate checking
28        // if the given number is odd
29      ind = part(a2, std::size(a2), /* ... */);
30      std::cout << "ind = " << ind << " ";
31      printTab(a2,std::size(a2));
32
33      std::string a3[] = {"Ala", "Ula", "Ela", "Ola", "Maja"};
34      std::string mn = "Bea";
35      std::string mx = "Sue";
36        // lambda as argument: a predicate checking
37        // if the given name is in the range [mn,mx]
38      ind = part(a3, std::size(a3), /* ... */);
39      std::cout << "ind = " << ind << " ";
40      printTab(a3,std::size(a3));
41
42      constexpr size_t DIM = 500000;
43      int* a4 = new int[DIM];
44      srand(unsigned(time(0)));
45      for (size_t i = 0; i < DIM; ++i) a4[i] = rand()%21+1;
46        // lambda as argument: a predicate checking
47        // if the given number is divisible by 7
48      ind = part(a4, DIM, /* ... */);
49      std::cout << "ind = " << ind << std::endl;
50      delete [] a4;
51  }
```

Fragment w liniach 42-43 służy do wygenerowania wartości do zainicjowania tablicy a4 (wszystkie wartości będą pochodzić z przedziału $[1, 21]$).

Operację rozdzielania elementów należy przeprowadzić w *jednej* pojedynczej pętli (bez pętli zagnieżdżonych), inaczej wywołanie z linii 46 dla tablicy o wymiarze pół miliona, trwałoby zbyt długo; przy poprawnej implementacji wykonanie powinno być praktycznie natychmiastowe. W funkcji **part** *nie* wolno tworzyć żadnych pomocniczych tablic!

Przykładowy wynik programu mógłby wyglądać tak:

```
ind = 3 [ 2 4 6 1 5 3 ]
ind = 3 [ 1 3 5 4 2 6 ]
ind = 3 [ Ela Ola Maja Ula Ala ]
ind = 71594
```
