# PJC

Ćwiczenia 04

**Zadanie 1**

Stwórz dwia szablony funkcji zwracające liczbę oraz przyjmujące dwa argumenty:

- funkcje add, której zadaniem będzie zwrócenie sumy argumentów
- funkcje sub, której zadaniem będzie zwrócenie różnicy argumentów

Przetestuj obie funkcje na kilku typach prostych (np. float, double, int, char)

**Zadanie 2**

Napisz szablon funkcji, która w podanej tablicy elementów dowolnego typu (dla którego ma sens porównywanie za pomocą operatora '<'), wyszukuje i zwraca wskaźnik największego elementu tablicy. Przetestuj funkcję dla tablic typu int[], double[] i string[].

**Zadanie 3**

Napisz program, w którym stworzysz wektor przechowujący liczby całkowite od 1 do 10. Następnie wypisz wszystkie elementy tego wektora. Stwórz funkcję przyjmującą jako parametr odnośnik do wektora przechowującego liczby całkowite, której zadaniem będzie przemieszanie zawartości wektora. Wykorzystaj do tego liczby pseudolosowe z wykorzystaniem rand() oraz srand().

**Zadanie 4**

Napisz szablon funkcji smallSum3, która pobiera wektor elementów typu całkowitego (zakładamy, że jego wymiar nie jest mniejszy od 3) a zwraca trzyelementowy wektor typu vector<size_t> zawierający indeksy trzech elementów wektora takich, że ich suma jest najbliższa wartości zerowej. Do obliczania wartości bezwzględnej (modułu) liczby całkowitej możesz wykorzystać funkcję abs (z nagłówka *cmath*). Na przykład program

```
#include <iostream>
    #include <cmath>
    #include <vector>

    template <typename T> std::vector<size_t>
    smallSum3(std::vector<T>& a) { // ...
    }
```

```
int main () {
      std::vector<int> a{2, -13, 3, 6, 4, 5, -14, 1, -15};
      auto r = smallSum3(a);

      std::printf("Sum=%d for elements " "a[%u]=%d, a[%u]=%d,
               a[%u]=%d\n", a[r[0]]+a[r[1]]+a[r[2]], r[0], a[r[0]], r[1],
               a[r[1]], r[2], a[r[2]]);
 }
```

powinien wypisać

```
Sum=-2 for elements a[1]=-13, a[3]=6, a[5]=5
```

Może się zdarzyć, że jest kilka rozwiązań — funkcja zwraca wtedy którekolwiek z nich.

**Zadanie 5**

Stwórz trzy szablony funkcji:

- Funkcja filtrująca

  ```
  template <typename T, typename FunType>
  vector<T> filter(const vector<T>& v, FunType p);
  ```

  która pobiera wektor v i funkcję (predykat) p pobierającą daną typu, jakiego są elementy wektora a zwracającą bool. Funkcja filter zwraca wektor tego samego typu co v i zawierający tylko te elementy wektora v, dla których predykat p zwraca true.

- Funkcja transformująca i filtrująca

  ```
  template <typename T, typename FunType1, typename
  FunType2>
  vector<T> transfilt(vector<T>& v, FunType1 t, FunType2 p);
  ```

  która pobiera wektor, funkcję transformującą t i predykat p. Wektor v (przesłany przez referencję) jest modyfikowany w ten sposób, że każdy jego element zastąpiony jest wynikiem działania na ten element funkcji transformującej. Funkcja transfilt zwraca wektor tego samego typu co v zawierający tylko te elementy przetransformowanego wektora v, dla których predykat p zwraca true.

- Funkcja drukująca

  ```
  template <typename T>
  void printVec(const vector<T>& v) {
  ```

  która pobiera wektor i drukuje w jednym wierszu, ujętym w nawiasy kwadratowe, jego elementy oddzielone znakami odstępu.

W wywołaniach funkcji filter i transfilt argumenty funkcyjne powinny być lambdami zdefiniowanymi bezpośrednio na liście argumentów. Jeśli w następującym programie:

```
#include <cmath>
#include <iostream>
#include <functional>
#include <vector>
using std::vector; using
std::function;

template <typename T, typename FunType> vector<T>
filter(const vector<T>& v, FunType p) { // ... }

template <typename T, typename FunType1, typename
FunType2>
vector<T> transfilt(vector<T>& v, FunType1 t, FunType2 p) { // ... }

template <typename T>
void printVec(const vector<T>& v) { // ... }

int main() { vector<int> v{1, -3, 4, -2, 6, -8, 5}; printVec(v);
    vector<int> r = filter(v, /* lambda_1 */); printVec(r);
    vector<int> s = filter(v, /* lambda_2 */); printVec(s);

    vector<double> w{1.5, -3.1, 4.0, -2.0, 6.3}; printVec(w);
    double mn = -0.5, mx = 0.5;
    vector<double> d = transfilt(w, /* lambda_3*/, /*
    lambda_4*/); printVec(w); printVec(d);
}
```

lambdy:

- lambda_1 — zwraca true dla liczb parzystych;
- lambda_2 — zwraca true dla liczb dodatnich;
- lambda_3 — zwraca sinus argumentu (std::sin z nagłówka *cmath*);
- lambda_4 — zwraca true dla liczb z zkresu [$mn$,$mx$], to powinien on wypisać

```
[ 1 -3 4 -2 6 -8 5 ]
[ 4 -2 6 -8 ]
[ 1 4 6 5 ]
[ 1.5 -3.1 4 -2 6.3 ]
[ 0.997495 -0.0415807 -0.756802 -0.909297 0.0168139 ]
[ -0.0415807 0.0168139 ]
```

Uwaga: nie używaj żadnych dodatkowych funkcji z biblioteki standardowej.