Zadanie 1

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

Zadanie 2

Napisz szablon funkcji *rekurencyjnej*, która dla podanej tablicy elementów dowolnego typu dla którego określone jest działanie operatora '<', zwraca wartość jej największego elementu. *Nie* używaj pętli ani żadnych pomocniczych tablic, kolekcji czy **string**ów! Przetestuj szablon dla tablic typu **int[]**, **double[]** i **string[]**.

Na przykład następujący program

download *SzabFunMaxRekur.cpp*

```
#include <iostream>
#include <string>

template<typename T>
T getMaxRecur(const T* arr, size_t size) {
    // ...
}

int main() {
    using std::cout; using std::endl; using std::string;

    double ad[]{2.5, 9.1, 4.5, -1.0};
    cout << "ad: " << getMaxRecur(ad,4) << "; ";

    int    ai[]{2, 11, 5, 1, 9};
    cout << "ai: " << getMaxRecur(ai,5) << "; ";

    string as[]{"Ala", "Ela", "Ula", "Ola"};
    cout << "as: " << getMaxRecur(as,4) << endl;
}
```

powinien wydrukować

```
ad: 9.1; ai: 11; as: Ula
```

Zadanie 3

Dla tablicy (wektora) nazwijmy *piwotem* element który jest niemniejszy od dowolnego elementu poprzedzającego (o niższym indeksie) i jednocześnie niewiększy od dowolnego z następujących po nim elementów (o wyższym indeksie).

Napisz i przetestuj następujące *szablony* funkcji (parametryzowane typem **T**):

- `void getPivots(const T* arr, int* info, size_t size);`
  pobiera tablicę arr o wymiarze size oraz tablicę intów info o takim samym rozmiarze; funkcja wypełnia tablicę info jedynkami na pozycjach odpowiadających piwotom w tablicy arr a zerami na pozostałych pozycjach.
- `std::vector<int> getPivots(std::vector<T>& v);`
  działa jak funkcja poprzednia, ale zamiast tablicy arr pobiera (przez referencję) wektor elementów typu T, a zamiast tablicy info zwraca tę samą informację w postaci wektora intów.
- `void print(const T* arr, const int* info, size_t size);`
  drukuje wyniki, w postaci pokazanej w przykładzie poniżej.

Uwaga: Obie funkcje powinny mieć liniową złożoność obliczeniową, a więc w ich treści nie może być pętli zagnieżdżonych.

Następujacy program, po zdefiniowaniu wszystkich szablonów funkcji

download *PivotElems.cpp*

```cpp
#include <iostream>
#include <string>
#include <vector>

template <typename T>
void getPivots(const T* arr, int* info, size_t size) {
    // ...
}

template <typename T>
std::vector<int> getPivots(std::vector<T>& v) {
    // ...
}

template <typename T>
void print(const T* arr, const int* info, size_t size) {
    // ...
}

int main() {
    int a[] = {1, 2, 1, 2, 2, 5};
    constexpr size_t size = std::size(a);
    int info[size];
    getPivots(a, info, size);
    print(a, info, size);

    std::vector<std::string> v{"A", "B", "A", "B", "B", "E"};
    std::vector<int> res = getPivots(v);
    print(v.data(), res.data(), v.size());
}
```

powinien wypisać

```
[ 0:1 3:2 4:2 5:5 ]
[ 0:A 3:B 4:B 5:E ]
```

gdzie liczba przed średnikiem to indeks odpowiadający piwotowi, a za średnikiem wartość tego elementu.

Zadanie 4

Przeprowadzamy ankietę na pewien temat. Wynik pojedynczej ankiety (uzyskanej od jednego respondenta) zawiera następujące informacje, które należy zakodować w *jednej* zmiennej typu **unsigned short** (można założyć, że ma ona 2 bajty, czyli 16 bitów):

1. płeć — 1 bit, bo 2 możliwości (kobieta, mężczyzna), kodowane jako 0 lub 1;
2. stan cywilny — 2 bity, bo 4 możliwości (panna/kawaler, mężatka/żonaty, rozwódka/rozwodnik, wdowa/wdowiec) kodowane jako 0, 1, 2 lub 3;
3. grupa wiekowa — 2 bity, bo 4 możliwości (np. 18-30, 31-45, 46-60, 60+) kodowane jako 0, 1, 2 lub 3;
4. wykształcenie — 2 bity, bo 4 możliwości (np. podstawowe, średnie, licencjat, magister+) kodowane jako 0, 1, 2 lub 3;
5. miejsce zamieszkania — 2 bity, bo 4 możliwości (np. wieś, miasto do 50 tys., miasto 50-400 tys., miasto ponad 400 tysięcy mieszkańców) kodowane jako 0, 1, 2 lub 3;
6. region kraju — 4 bity, bo (przypuśćmy) jest 16 regionów ponumerowanych od 0 do 15;
7. odpowiedź na pytanie ankietera – 3 bity, bo w ankiecie (przypuśćmy) było 8 możliwych odpowiedzi, ponumerowanych od 0 do 7.

Napisz funkcję

```
unsigned short koduj(int plec, int stan_cyw,
                     int grupa_wiek, int edu,
                     int zam, int region,
                     int odp);
```

która pobiera 7 liczb (jak wyżej), koduje informacje w nich zawarte w *jednej* zmiennej typu **unsigned short** i zwraca tę jedną liczbę do funkcji wywołującej.
Napisz również funkcję

```
void info(unsigned short kod);
```

która pobiera jeden argument typu **unsigned short** zawierający informacje o jednej ankiecie i wypisuje te informacje, np. w formie:

```
plec:           0
stan cywilny:   3
grupa wiekowa:  2
wyksztalcenie:  3
miejsce zam.:   0
region:        12
odpowiedz:      6
```

Nie używaj żadnych narzędzi z biblioteki standardowej (innych niż te z *iostream* do wypisywania wyników). Zamiast **unsigned short** możesz użyć **uint16_t** (chociaż na większości platform są to synonimy).
