AiSD Laboratorium 2

Na tym laboratorium zadania dotyczą tablic. Ich treść niekoniecznie jest związana z treścią projektu 1.

UWAGA:

Niektóre z funkcji, które trzeba zaimplementować poniżej są już zdefiniowane w bibliotece standardowej w nagłówku `<algorithm>`. Celem zadań jest samodzielne zmierzenie się z problemem. Po wszystkim zachęcam do zapoznania się funkcjami bibliotecznymi.

Zadanie 1.

Zaimplementować funkcję wyświetlającą na ekranie zawartość tablicy A, przechowującej elementy typu T. Rodzaj użytej tablicy dowolny. W zależności od wybranego rodzaju tablicy, postać funkcji musi być następująca:

- w przypadku typu wbudowanego

```
void print_legacy_array(const T* A, size_t N);
```

- w przypadku typu `std::array`

```
template<typename T, size_t N>
void print_std_array(const std::array<T,N>& A);
```

- w przypadku typu `std::vector`

```
template<typename T>
void print_std_vector(const std::vector<T>& A);
```

Aby nie korzystać z szablonów można przyjąć, że T=`short`. Konkretny format wyświetlania jest dowolny.

Zadanie 2.

Zaimplementować funkcję kopiującą tablicę A rozmiaru N_A do tablicy B rozmiaru N_B. Obie tablice przechowują elementy typu T. W przypadku gdy:

- N_A <= N_B pozostałe pola tablicy B zachowują wartości oryginalne
- N_A > N_B nadmiarowe pola tablicy A są odrzucane

Rodzaj użytej tablicy dowolny, można założyć, że obie są tego samego typu. W zależności od wybranego rodzaju tablicy, postać funkcji musi być następujaca:

- w przypadku typu wbudowanego

```
void copy_legacy_array(const T* A, size_t N_A, T* B, size_t N_B);
```

- w przypadku typu std::array

```
template<typename T, size_t N_A, size_t N_B>
void copy_std_array(const std::array<T,N_A>& A, std::array<T,N_B>& B);
```

- w przypadku typu std::vector

```
template<typename T>
void copy_std_vector(const std::vector<T>& A, std::vector<T>& B);
```

Aby nie korzystać z szablonów można przyjąć, że T=short.

Zadanie 2 a.

Zaimplementować kopiowanie w taki sposób, aby możliwe było podanie indeksu w tablicy B od którego mają być wstawione elementy tablicy A np.:

```
void copy_legacy_array(const T* A, size_t N_A, T* B, size_t N_B, size_t offset);
```

Zadanie 2 b.

Zaimplementować kopiowanie pomiędzy tablicami różnych typów np.: wbudowaną i std::array.

Zadanie 3.

Zaimplementować funkcję zapisującą do pliku zawartość tablicy A przechowującej elementy typu T. Rodzaj użytej tablicy dowolny. W zależności od wybranego rodzaju tablicy, postać funkcji musi być następujaca:

- w przypadku typu wbudowanego

```
void write_legacy_array(std::string filename, const T* A, size_t N);
```

- w przypadku typu `std::array`

```
template<typename T, size_t N>
void write_std_array(std::string filename, const std::array<T,N>& A);
```

- w przypadku typu `std::vector`

```
template<typename T>
void write_std_vector(std::string filename, const std::vector<T>& A);
```

Aby nie korzystać z szablonów można przyjąć, że T=`short`. Konkretny format pliku jest dowolny.

Zadanie 4.

Zaimplementować funkcję zliczającą ilość wystąpień elementu w tablicy.

Zadanie 5.

Zaimplementować funkcję odwracającą kolejność elementów w tablicy. Zwrócić uwagę na zminimalizowanie czasu działania i dodatkowej pamięci.

Zadanie 6.

Zaimplementować funkcję zwracającą największy i najmniejszy element tablicy.

Zadanie 7.

Zaimplementować funkcje, która wypełni tablicę kolejnym wyrazami ciągu arytmetycznego.

Zadanie 8.

Niech dane będą tablica `A` oraz funkcja `T foo(T)`. Zaimplementować funkcję, która przekształci każdy element tablicy za pomocą `foo`. Wynik przekształcenia elementu `A[i]` ma być zapisany ponownie w `A[i]`.