AiSD Laboratorium 1

Jest to zestaw zadań ,,na rozgrzewkę''. Rozwiązanie (lub nie) tych zadań nie jest elementem oceny.

Ich celem jest przyzwyczajnie Państwa do formatu w jakim będą prezentowane zadania, ubocznym efektem powinno być przypomnienie sobie jak się pisze programy, przygotowanie IDE etc...

Wymagania wstepne:

- Do pisania i uruchamiania programów bedzie używane Visual Studio. Proszę przypomnieć sobie jak w tym środowisku:
  - tworzy się projekt
  - uruchamia się program
  - gdzie sa umieszczone pliki projektu
- Programy będa pisane w języku C++ (formalnie standard C++17 lub C++20). Podstawy programowania w tym języku były na innym przedmiocie. Na potrzeby tych zajęć będą potrzebne
  - podstawowe konstrukcje składniowe
  - obsługa I/O
  - praca z łańcuchami znakowymi
  - podstawy programowania obiektowego
  - mierzenie czasu
  - generowanie liczb pseudolosowych
  - Proszę zapoznać się z plikiem `AiSD_lab_01_COMMENTS.pdf` oraz zawartością archiwum `AiSD_lab_01_demos.zip`

Zadanie 0.

W Visual Studio utworzyć nowy projekt w C++ i napsac w nim program ,,Hello World".

Zadanie 1.

Napisać program wyświetlający na ekranie poniższy tekst **dokładnie** w tym formacie:

```
Hello World!
AiSD
   5   8  11  14  17  20  23  26  29  32  35  38  41  44  47  50  53  56  59  62
  65  68  71  74  77  80  83  86  89  92  95  98 101 104 107 110 113 116 119 122
 125 128 131 134 137 140 143 146 149 152 155 158 161 164 167 170 173 176 179 182
 185 188 191 194 197 200 203 206 209 212 215 218 221 224 227 230 233 236 239 242
 245 248 251 254 257 260 263 266 269 272 275 278 281 284 287 290 293 296 299 302
```

Zadanie 2.

Wbudowany operator `sizeof(T)` zwraca rozmiar w bajtach typu lub zmiennej `T`. Na przykład:

```
char a {'A'};

std::cout "sizeof(char): " << sizeof(char) << '\n';
std::cout "sizeof(a): " << sizeof(a) << '\n';
```

da w wyniku:

```
sizeof(char): 1
sizeof(a): 1
```

Napisać program spełniajacy poniższe wymagania:

Wymagania podstawowe

- wypisze na ekranie nazwy wbudowanych typów znakowych, całkowitoliczbowych i zmiennoprzecinkowych wraz z ich rozmiarami w bajtach i bitach.
- zapisze te same dane do pliku `size_of_builtin_types.txt`

Wymagania dodatkowe

- wykona te same czynności dla wszystkich typów wbudowanych, w szczególności dla
  - typu `bool`
  - wskaźników do różnych typów
  - referencji do róznych typów
- wykona te same czynności dla typów/zmiennych

```
struct A{
    char c;
    int i;
    long l;
};

// w struct B; zmieniona została kolejność deklarowania pól,
// może to prowadzć do zmiany rozmiaru
struct B{
    int i;
    long l;
    char c;
};

struct C{
    char* c;
    int* i;
    long* l;
};
```

Listę typów wbudowanych można znaleźć tutaj.

Zadanie 3.

W nagłówku `<limits>` zdefiniowane jest wiele pomocniczych funkcji opisujących właściwości typów wbudowanych, np.:

- `std::numeric_limits<T>::lowest()`
- `std::numeric_limits<T>::min()`
- `std::numeric_limits<T>::max()`
- `std::numeric_limits<T>::digits10`

Korzystając z tych funkcji, napisac program który

Wymagania podstawowe

- zapisuje do pliku `numeric_limits.txt` informacje o tym jak duże liczby mogą przechowywać typy wbudowane.

Wymagania dodatkowe

- zapisać do pliku w formie czytelnej tabelki. przykładowa tabelka:

```
type                 size       lowest()        min()                 max() digits10
-------------------------------------------------------------------------------------
char                    1           -128         -128                   127        2
signed char             1           -128         -128                   127        2
int                     4    -2147483648  -2147483648            2147483647        9
unsigned long long int  8              0            0  18446744073709551615       19
long double            16 -1.18973e+4932 3.3621e-4932         1.18973e+4932       18
-------------------------------------------------------------------------------------
```

- zapisać do pliku w formacie CSV

Zadanie 4.

1. Zapoznać się z:
   - wbudowanym typem tablicowym (tablicami w stylu C),
   - kontenerem std::array
   - kontenerem std::vector

2. Utworzyć tablicę `int A[]` zawierajacą elementy `{0,1,2,3,4,5,6,7,8,9}`. Wyjaśnić co się stanie się w wyniku wykonania kodu:

```
int a = A[2];
A[5] = 13;

A[-1] = -1;
int b = A[10]
```

3. Utworzyć tablicę `B` typu `std::array<int,10>`. Sprawdzić, co się stanie w wyniku wykonania kodu:

```
B[-1]=-1
int a = B[10]
B[10]=1

B.at(-1)=1
int b = B.at(10)
```

4. Utworzyć tablicę `C` typu `std::vector<int>` zawierajacą 10 elementów. Sprawdzić, co się stanie w wyniku wykonania kodu:

```
C[-1]=-1
int a = C[10]
C[10]=1

C.at(-1)=1
int b = C.at(10)
```

5. Wyświetlić na ekranie i zapisać do pliku położenie w pamięci elementów tablic `A`, `B` i `C` wraz z odpowiadającymi im elementami.

Projekt 0a - nieoceniany

Celem tego projektu jest zmierzenie czasu działania algorytmu. Funkcje, których czas działania będzie mierzony mają następujące deklaracje:

```
/*
 * zwraca sumę liczb od 0 do k
 * wynik jest obliczany poprzez iteracyjne dodawanie kolejnych elementów
 */
unsigned long long SumUpTo_addition(unsigned long long k);

/*
 * zwraca sumę liczb od 0 do k
 * wynik jest obliczany ze wzoru na sumę kolejnych liczb całkowitych
 */
unsigned long long SumUpTo_formula(unsigned long long k);
```

Wymagania podstawowe

- Zaimplementować funkcje `SumUpTo_addition` oraz `SumUpTo_formula`.
- Określić maksymalne `k` dla którego funkcje zwrócą poprawny wynik.
- Wykonać eksperyment polegający na:
  - uruchomieniu obu funkcji dla `1000` różnych `k` z zakresu `1..max_k`
  - zmierzeniu czasu działania każdego uruchomienia
  - wybór `k` jest dowolny, dla lepszego efektu lepiej jest wybrać je tak, aby nie rosły równomiernie, np.: dla `k<1000` krok co `100`, dla `k<10000` krok co `1000` itd.
  - aby otrzymać lepsze oszacowanie czasu działania najlepiej jest wywołać funkcję kilkukrotnie dla tego samego `k` i uśrednić otrzymane wyniki
- Otrzymane czasy zapisać do pliku w formacie `csv`. Każdy wiersz w pliku musi być w formacie

```
funkcja, k, suma, czas działania
```

Wymagania dodatkowe

- rozszerzyć eksperyment o funkcję która:
  - wyświetla na ekranie kolejne liczby od `1` do `k`
- rozszerzyć eksperyment o funkcję która
  - sumuje kolejne liczby od `1` do `k`
  - wyświetla na ekranie sumy częściowe podczas ich obliczania
- (*) rozszerzyć eksperyment o funkcje z biblioteki standardowej zdefiniowane w nagłówku `<algorithms>`:
  - `std::accumulate`
  - `std::partial_sum`
- (*niekoniecznie w C++*) określić teoretycznie i empirycznie złożoność obliczeniową zbadanych algorytmów, np.: poprzez narysowanie wykresu

Projekt 0b - nieoceniany

To zadanie zdecydowanie lepiej wykonać w Excelu, Mathematice lub R, ale bez problemu można w C++

Wymagania podstawowe

- Załóżmy, że w czasie jednej sekundy pewien komputer wykonuje 1'500'170 operacji.
- Na tym komputerze uruchomiono algorytmy:
  - A1: o czasie działania `1000 log(n)`
  - A2: o czasie działania `100 n`
  - A3: o czasie działania `10 n log(n)`
  - A4: o czasie działania `n^2`
  - A5: o czasie działania `n^3`
  - A6: o czasie działania `2^n`
- Wyznaczyć czas działania tych algorytmów dla zbiorów danych o rozmiarach:
  - 1, 5, 10, 25, 50, 100, 102, 103, 110, 200, 250, 500, 1000
- Otrzymane wyniki zapisać do pliku w formacie CSV.

Wymagania dodatkowe

- Wyznaczyć dokładne wartości `n` dla których poszczególne algorytmy się prześcigają.