

## **Zadanie 1.** Przygotowanie środowiska i narzędzi, **klasa do rejestracji informacji diagnostycznych.**

Zakres zagadnień do przypomnienia (przygotowania) na potrzeby „zadania 1” (02.03.2023)

- Podstawowy projekt konsolowy w Code::Blocks (MinGW 64-bit), dołączanie plików klas/bibliotek do projektu
- Praca z plikami tekstowymi (utworzenie, odczyt, zapis, dopisywanie treści)
- Podstawowe typy (liczby całkowite, liczby zmiennoprzecinkowe), `std::string`, `std::vector`
- Konwersja typów:
  - `int <-> string` (również `int -> string` w postaci HEX)
  - `double <-> string` (ze zmienną precyzją)
- Dyrektywy preprocesora
  - `include`
  - `define`
  - `ifdef, ifndef, else`
- Klasy statyczne

# Programowanie i wizualizacja interfejsów (1500-DIIB6PWI)

Cel „zadania 1”:

Napisać klasę statyczną (dołączaną później do kolejnych programów), która umożliwi rejestrowanie informacji diagnostycznych w pliku tekstowym. Proponowany sposób wykorzystania:

```
if (cos_tam == 0) {  
    DIAG_BUF("time_handler_ls not allowed") // do bufora  
    DIAG_BUF("jakis kolejny tekst") // do bufora  
    DIAG_OUT // przeniesienie zawartości bufora do pliku  
    return TRUE;  
}  
  
if (cos_tam2 > 0) {  
    DIAG(_m_d2s(zmienna_double,4) + " jakis tekst " + _m_i2s(zmienna_int)) // konwersja double z 4 miejscami po przecinku + konwersja int  
    return TRUE;  
}
```

Założenia:

Jeżeli jest zdefiniowana stała (np. **DIAG\_ENABLE**) to wszystkie polecenia (makra) **DIAG\_xxx** działają.

Jeżeli ta stała nie jest zdefiniowana, rejestracja zapisów diagnostycznych jest wyłączona (makra istnieją ale są „puste”).

Dwa sposoby rejestracji:

1: Informacje diagnostyczne są zbierane w buforze (np. polecenie **DIAG\_BUF**), bufor jest zapisywany do pliku osobnym poleceniem (np. **DIAG\_OUT**). Działa szybciej, ale można stracić informacje jeśli program nie zdąży zapisać bufora do pliku.

2: Informacje diagnostyczne są zapisywane w buforze (w buforze już mogła być jakaś treść), zawartość bufora jest natychmiast przepisywana do pliku (polecenie **DIAG**). Działa wolniej ale jest mniejsze prawdopodobieństwo, że informacja zostanie stracona.

## Programowanie i wizualizacja interfejsów (1500-DIIB6PWI)

### Opcja A:

Wprowadzenie wcięć w pliku diagnostycznym (realizowanych np. znakiem tabulacji):  
Polecenie (makro) **DIAG\_INDP** – INDentacja Plus w parze z **DIAG\_INDM** (INDentacja Minus)

```
function F1 () {  
    DIAG_INDP  
    ...  
    ...  
    DIAG_INDM  
}
```

### Opcja B:

Wprowadzenie funkcjonalności pozwalającej na globalne definiowanie stopnia diagnostyki, np. stała **DIAG\_LEV** (LEVel)  
np. o wartościach 1 (najważniejsze), 2, 3, 4 (najmniej ważne).

Zapisy diagnostyczne będą wymagały dodania parametru - stopnia ważności, np.: **DIAG\_BUF**(1, "bardzo wazna informacja")

Przykład: jeżeli **DIAG\_LEV** = 2, to wszystkie wpisy o ważności 3,4.. nie będą zapisywane.

### Opcja B2:

Polecenie **DIAG\_LEV\_TEMPSET**(X) tymczasowo zmieniające **DIAG\_LEV**, np. na początku funkcji, kiedy chcemy tymczasowo głębiej wejrzeć w sposób działania funkcji. Powinno działać w parze z poleceniem przywracającym **DIAG\_LEV\_RESTORE**.

Ponieważ diagnozowana funkcja może wywoływać funkcje z inną wartością **DIAG\_LEV\_TEMPSET**, to dotychczasową wartość (do której będziemy wracać **DIAG\_LEV\_RESTORE**) należy odkładać na stos (LIFO).

# Programowanie i wizualizacja interfejsów (1500-DIIB6PWI)

Później będzie potrzebne:

Wprowadzenie kilku analogicznych klas zapisujących dane diagnostyczne (każda do swojego pliku).

Każda klasa służy do diagnostyki innego wątku.

Do każdego wpisu należy automatycznie dodać znacznik czasowy, żeby później można było złożyć jeden plik diagnostyczny.

Potrzebny będzie program do „składania” i przeglądania plików diagnostycznych

– warto opracować standard.

## Opcja C

Wspólne pliki headera i opisu klasy: **diagx.hpp** + **diagx.cpp**, zmieniają się tylko indeksy w indywidualnych plikach diag1.hpp + diag1.cpp, diag2.hpp + diag2.cpp itd..

diag1.hpp:

```
#ifndef DIAG_INDEX
#define DIAG_INDEX
#endif

// odtad zmieniamy
#ifndef _C_DIAG1_HPP
#define _C_DIAG1_HPP

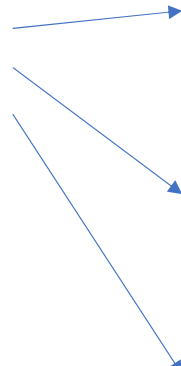
#define DIAG_INDEX 1 // identyfikator odrozniajacy

// dalej nie trzeba nic zmieniac
#include "diagx.hpp" // wczytaj czesc wspolna
#endif
```

diag1.cpp:

```
// nagłówek z odpowiednim indeksem
#include "diag1.hpp"

// wczytaj czesc wspolna
#include "elogx.cpp"
```



```
class diag {...} // diag.hpp

void diag::o() {} // diag.cpp
void mlog::init() {}
..

class diag2 {...} // diag2.hpp

void diag2::o() {} // diag2.cpp
void mlog2::init() {}
..

class diag3 {...} // diag3.hpp

void diag3::o() {} // diag3.cpp
void mlog3::init() {}
..
```