

✿ TiDA 02: Tablice dynamiczne i statyczne, stos i sarta, testy funkcjonalne i wydajnościowe

🎯 Cel zajęć

- Na tych zajęciach: - Poznasz różnicę między **tablicą statyczną** a **dynamiczną**.
- Dowiesz się, czym są **stos (stack)** i **sarta (heap)**.
 - Zrozumiesz, jak architektura procesora (**x86** vs **x64**) wpływa na wydajność programu.
 - Dowiesz się, jak odróżniać **testy funkcjonalne** i **testy wydajnościowe**.
 - Rozszerzenie poprzedniego programu o nowe funkcje pomiarowe i testy poprawności działania.

🧠 CZĘŚĆ TEORETYCZNA

💎 Stos (stack) i sarta (heap)

Obszar	Co tam trafia	Cechy
Stos (stack)	zmienne lokalne, tablice statyczne	szybki dostęp, automatyczne zwalnianie pamięci po zakończeniu funkcji
Sarta (heap)	dane dynamiczne (tworzone przez new, malloc)	programista sam zarządza pamięcią (delete, free)

Tablica statyczna

```
int tablicaStatyczna[1000];
```

Tworzona **na stosie** – jej rozmiar musi być znany w czasie kompilacji. Po zakończeniu funkcji pamięć zostaje **automatycznie zwolniona**.

💎 Tablica dynamiczna

Tworzona **na sterwie** – rozmiar może być określony **w czasie działania programu**.

W języku C:

```
int* tablicaDynamiczna = (int*)malloc(rozmiar * sizeof(int));  
...  
free(tablicaDynamiczna);
```

W języku C++:

```
int* tablicaDynamiczna = new int[rozmiar];  
...  
delete[] tablicaDynamiczna;
```

Ważne: jeśli zapomnisz o free() lub delete[], pamięć **nie zostanie zwolniona** → tzw. **wyciek pamięci (memory leak)**.

◆ Sprawdzanie typu tablicy

- jeśli użyto `new` lub `malloc` → tablica dynamiczna (na stercie),
- jeśli `int t[100];` → tablica statyczna (na stosie).

Jeśli tablica statyczna będzie zbyt duża, program zakończy się błędem **Stack overflow**.

◆ Testy funkcjonalne vs wydajnościowe

Typ testu	Co sprawdza	Przykład
Funkcjonalny	Czy program działa poprawnie	Czy tablica istnieje, jest poprawnie wypełniona i posortowana
Wydajnościowy	Jak szybko program działa	Pomiar czasu sortowania dla różnych rozmiarów tablic

◆ Architektura x86 vs x64

- **x86** – 32-bitowa, krótsze rejestry ogólnego przeznaczenia (4 bajty)
 - **x64** – 64-bitowa, dłuższe rejestry ogólnego przeznaczenia (8 bajtów), większa przestrzeń adresowa
Kod 64-bitowy często działa szybciej, ale nie zawsze — dla małych danych może być odwrotnie.
-

CZĘŚĆ PRAKTYCZNA

◆ Tworzenie tablicy dynamicznej i statycznej

```
int* utworzTablice(int rozmiar, bool dynamiczna) {
    if (dynamiczna) {
        cout << "Tworze tablice dynamiczna..." << endl;
        return new int[rozmiar];
    } else {
        cout << "Tworze tablice statyczna..." << endl;
        static int tablica[100000];
        return tablica;
    }
}
```

◆ Testy funkcjonalne

```
bool czyTablicaPoprawna(int* tablica, int rozmiar, int MIN, int MAX) {
    if (!tablica) return false;
    for (int i = 0; i < rozmiar; i++) {
        if (tablica[i] < MIN || tablica[i] > MAX) return false;
    }
    return true;
}

bool czyTablicaPosortowana(int* tablica, int rozmiar) {
    for (int i = 1; i < rozmiar; i++) {
        if (tablica[i - 1] > tablica[i]) return false;
    }
    return true;
}

bool testFunkcjonalny(int* tablica, int rozmiar, int MIN, int MAX) {
    return czyTablicaPoprawna(tablica, rozmiar, MIN, MAX) &&
        czyTablicaPosortowana(tablica, rozmiar);
}
```

Pomiar czasu

```
#include <chrono>
using namespace std::chrono;

void testSortowania(int rozmiar, bool dynamiczna, int MIN, int MAX) {
    int* tablica = utworzTablice(rozmiar, dynamiczna);

    srand(time(nullptr));
    for (int i = 0; i < rozmiar; i++)
        tablica[i] = rand() % (MAX + 1);

    auto start = high_resolution_clock::now();
    sort_bubble(tablica, rozmiar);
    auto end = high_resolution_clock::now();

    long long czas = duration_cast<microseconds>(end - start).count();
    cout << "Czas sortowania: " << czas << " mikrosekund" << endl;

    if (testFunkcjonalny(tablica, rozmiar, MIN, MAX))
        cout << "✅ Test funkcjonalny zaliczony" << endl;
    else
        cout << "❌ Test funkcjonalny niezaliczony" << endl;

    if (dynamiczna) delete[] tablica;
}
```

ZADANIA

1. Uruchom program i sprawdź różnicę w czasie między tablicą dynamiczną a statyczną.
2. Zwiększ rozmiar tablicy statycznej aż do błędu *Stack overflow* i zanotuj wynik.
3. Zmierz czasy sortowania dla różnych rozmiarów (1000, 10000, 100000 elementów).
4. Uruchom program w wersji **x86** i **x64** – porównaj czasy.
5. Napisz raport z wynikami:
 - Który typ tablicy działał szybciej?
 - Przy jakim rozmiarze wystąpił błąd stosu?
 - Czy kod x64 był szybszy?
 - Jakie testy funkcjonalne wykonałeś?