


Sprawozdanie z programów z listy drugiej			
 <p>Wrocław University of Science and Technology</p>	Student: <i>Jakub Król</i> 226269	Data Laboratoriów: 22.03.2017r. g.16:30	Prowadzący: <i>Dr inż. Krzysztof</i> <i>Halawa</i>
		Wykonano: 28.03.2017r.	
		Grupa laboratoryjna: E02-18p	Ocena:

Wstęp:

Programy głównie opierały się na **porównaniu czasu różnych implementacji**. Autor w dalszej części dokumentu będzie prezentował swoje programy w **języku C/C++**. Do programów obiektowych został stworzony plik makefile do kompilacji.

Programy obiektowe zostały napisane zgodnie ze sztuką hermetyzacji danych oraz używają szablonów.

W kodzie nie ma wielu komentarzy, jednak wszelakie **funkcje** oraz **zmienne są nazywane zgodnie z zastosowaniem** w języku angielskim, tak aby **kod był czytelny i przejrzysty**.

Wnioski nie są zbierane pod koniec dokumentu, są one wypisywane przy konkretnym zagadnieniu.

W programach są używane wskaźniki oraz zmienne dynamiczne, aby sprawdzić czy nie następują wycieki pamięci posłużono się programem valgrind z opcja leak-check=full.

Zostały wykonane zadania 1,3 oraz 4 na ocenę b.dobłą, zgodnie z poleceniem z dokumentu.

UWAGA1:

Na ocenę **dostateczną** należy wykonać zadania 1 i 2, na ocenę **dobrą** należy wykonać zadania 1, 2 i 3, a na ocenę **bardzo dobrą** należy wykonać zadanie 1 lub 2 oraz zadania 3 i 4. Do zadania 3 należy napisać krótkie sprawozdanie według wytycznych na końcu listy zadań.

Zadanie 1 & 3

1. Należy zaimplementować stos przechowujący elementy określonego typu (np.: int, float, itp.). Należy napisać funkcje wykonujące podstawowe operacje na stosie (dodawanie i usuwanie elementu, usuwanie wszystkich elementów oraz wyświetlanie zawartości stosu) z zastosowaniem:
 - (a) implementacji bazującej na tablicy
 - (b) implementacji w oparciu o bibliotekę STL.
3. Należy przeprowadzić eksperymenty polegające na zmierzeniu czasu działania operacji dodawania elementów na strukturach danych z zadania 1 i 2 w zależności od zastosowanej metody implementacyjnej. Struktury danych powinny być wypełniane losowymi danymi. Proszę o porównanie czasów działania dla implementacji bazującej na powiększanej tablicy (dwie strategie) i na liście (z poprzednich zajęć). Proszę przeprowadzić badania dla rozmiarów danych wejściowych równych 1 000, 10 000, 100 000 i 500 000.

Opis:

Powyższe dwa zadania zostały zawarte w jednym programie, jako pomoc najpierw napisano sam kontener – stos oparty na wskaźnikach oraz na tablicy, a następnie zebrano w **jeden program porównujący czasy implementacji opartej o:**

- Tablice
- Stos na wskaźnikach
- Stos z STL

W klasie stack_a – stos oparty o tablicę, umieszczono **dwie metody posługujące się dwoma różnymi strategiami** dodawania nowych elementów (poszerzania tablicy dynamicznej):

- **1 metoda:** powiększanie tablicy dynamicznej o 1 (optymalizacja pamięci)
- **2 metoda:** powiększanie tablicy dynamicznej dwukrotnie (optymalizacja obliczeń)

Nie zostało to ujęte w jednej metodzie, gdyż dodatkowy czas sprawdzenia (warunek if) **zwiększałby wymagany czas obliczeń.**

Kod programu:

https://github.com/Rexluu/PAMSI/tree/Final_versions/List_3/L3_Z1_Z3

Czysty stos na wskaźnikach:

https://github.com/Rexluu/PAMSI/tree/Final_versions/List_3/Stack_pointer

Testy:

Przeprowadzono najpierw mały test na 10 elementach z użyciem programu valgrind, czy nie ma żadnych wycieków pamięci.

```
==4939== Memcheck, a memory error detector
==4939== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4939== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
==4939== Command: ./L3_Z1
==4939==
Podaj ilość danych do testu: 10
```

Czas tablicy(stosu) powiększanej o 1: 0.005239[s]

Czas tablicy(stosu) powiększanej dwukrotnie: 0.00133[s]

Czas stosu z STL: 0.002161[s]

Czas własnego stosu: 0.001783[s]

```
==4939==
==4939== HEAP SUMMARY:
==4939==    in use at exit: 0 bytes in 0 blocks
==4939==   total heap usage: 29 allocs, 29 frees, 2,000 bytes allocated
==4939==
==4939== All heap blocks were freed -- no leaks are possible
==4939==
==4939== For counts of detected and suppressed errors, rerun with: -v
==4939== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Następnie przeprowadzono testy dla: 1; 10; 100; 10 000; 100 000; 500 000

Podaj̄ ilosc danych do testu: 1

Czas tablicy(stosu) powiekszanej o 1: 2e-06[s]

Czas tablicy(stosu) powiekszanej dwukrotnie: 1e-06[s]

Czas stosu z STL: 1e-06[s]

Czas własnego stosu: 3e-06[s]

Podaj̄ ilosc danych do testu: 10

Czas tablicy(stosu) powiekszanej o 1: 5e-06[s]

Czas tablicy(stosu) powiekszanej dwukrotnie: 1e-06[s]

Czas stosu z STL: 0[s]

Czas własnego stosu: 1e-06[s]

Podaj̄ ilosc danych do testu: 100

Czas tablicy(stosu) powiekszanej o 1: 0.000119[s]

Czas tablicy(stosu) powiekszanej dwukrotnie: 2.1e-05[s]

Czas stosu z STL: 1.1e-05[s]

Czas własnego stosu: 2.4e-05[s]

Podaj̄ ilosc danych do testu: 1000

Czas tablicy(stosu) powiekszanej o 1: 0.003069[s]

Czas tablicy(stosu) powiekszanej dwukrotnie: 1.7e-05[s]

Czas stosu z STL: 1.9e-05[s]

Czas własnego stosu: 3.5e-05[s]

Podaj̄ ilosc danych do testu: 10000

Czas tablicy(stosu) powiekszanej o 1: 0.143992[s]

Czas tablicy(stosu) powiekszanej dwukrotnie: 0.00012[s]

Czas stosu z STL: 0.000157[s]

Czas własnego stosu: 0.000356[s]

Podaj ilość danych do testu: 100000

Czas tablicy(stosu) powiększanej o 1: 16.1323[s]

Czas tablicy(stosu) powiększanej dwukrotnie: 0.001064[s]

Czas stosu z STL: 0.001287[s]

Czas własnego stosu: 0.003434[s]

Podaj ilość danych do testu: 500000

Czas tablicy(stosu) powiększanej o 1: 444.325[s]

Czas tablicy(stosu) powiększanej dwukrotnie: 0.005325[s]

Czas stosu z STL: 0.006632[s]

Czas własnego stosu: 0.018484[s]

Wnioski:

- Z powyższych testów jasno wynika że **implementacja** na tablicy dynamicznej **powiększanej inkrementacyjnie** jest **b. słabym, nieoptymalnym rozwiązaniem**.
- Zaskoczeniem okazuje się rozwiązanie **tablicy powiększanej dwukrotnie**, dodawanie elementów do takiej tablicy **jest najszybsze, jednak przy danych np. w ilości $2n+1$ pozostaje dużo nieużytków pamięci**, przez co to rozwiązanie pomimo że najszybsze jest **nieoptymalne pamięciowo**.
- Pozostałe dwa rozwiązania to stos z STL oraz stos napisany przez autora, oba te rozwiązania są nieznacznie wolniejsze od poprzedniego, jednak są optymalne pamięciowo. Z testów oraz z przedstawionych wyżej informacji, jasno wynika, iż są to najlepsze rozwiązania.
- **Przyczyną tego, iż stos napisany przez autora jest wolniejszy od stosu z STL jest prawdopodobnie fakt, że autor podzielił swój stos na dwie klasy. Przy każdym nowym dodanym elemencie jest dynamicznie alokowana nowa cała klasa zamiast jednej zmiennej. Jest to bardziej uniwersalne rozwiązanie, ale jak widać jest też wolniejsze – coś za coś. Autor wyciągnął wniosek na podstawie tego, że implementacja na tablicy powiększanej dwukrotnie najmniej razy alokuje pamięć, a testy pokazują że jest najszybsza.**
- Autor porównał także czasowo jego stos z vectorem z STL, okazało się że stos autora jest szybszy niż vector z STL.

Zadanie 4

Opis:

Program ten wykorzystuje stos autora, który został już sprawdzony pod względem wycieków, dlatego też w testach nie będzie to sprawdzane. Problem wieży Hanoi został rozwiązany rekurencyjnie.

Kod programu:

https://github.com/Rexluu/PAMSI/tree/Final_versions/List_3/L3_Z4

Testy:

Podaj wysokosc wiezy hanoi [0-inf]: 2

- - - - -

Palik 1

1

2

- - - - -

Palik 2

- - - - -

Palik 3

- - - - -

- - - - -

- - - - -

- - - - -

Palik 1

2

- - - - -

Palik 2

1

- - - - -

Palik 3

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -

- - - - -