


Sprawozdanie z programów z listy szóstej			
 <p>Wrocław University of Science and Technology</p>	<b>Student:</b>  <i>Jakub Król</i> 226269	<b>Data Laboratoriów:</b> 10.05.2017r. g.16:30  <b>Wykonano:</b> 14.05.2017r.	<b>Prowadzący:</b> <i>Dr inż. Krzysztof</i> <i>Halawa</i>
		<b>Grupa laboratoryjna:</b> E02-18p	<b>Ocena:</b>

## Wstęp:

Programy opierały się na tworzeniu abstrakcyjnych formatów (baz) danych. 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.**

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 opcją leak-check=full.**

**Programy nie korzystają z STL zgodnie z zaleceniami wykładowcy.**

# Program 1

1. Należy zaimplementować tablicę haszującą przechowującą  $N$  integerów z wykorzystaniem funkcji haszującej w postaci  $x\%N$ , gdzie  $x$  reprezentuje wprowadzaną wartość, a  $N$  jest wielkością tablicy haszującej.
  - (a) Użytkownik powinien mieć możliwość zdefiniowania ilości wprowadzanych elementów (należy zweryfikować, że jest to liczba pierwsza),
  - (b) wartości zapisywane do tablicy powinny być generowane losowo bez powtórzeń,
  - (c) należy zaimplementować trzy sposoby rozwiązywania kolizji:
    - i. linkowanie
    - ii. próbkowanie liniowe
    - iii. podwójne haszowanie z drugą funkcją haszującą w postaci  $q - x\%q$ , gdzie  $q < N$  jest liczbą pierwszą.
  - (d) należy zaprezentować działanie operacji dodawania, wyszukiwania oraz usuwania elementów z tablicy, za każdym razem wyświetlając ilość próbek potrzebnych do dodania i wyszukania elementów.

## Opis:

Należało wykonać hashowanie dodawanych elementów oraz rozwiązanie ich kolizji za pomocą trzech przypadków: linkowania, próbkowania liniowego oraz podwójnego hashowania. W przypadku próbkowania liniowego oraz podwójnego hashowania, program operuje na tablicach dynamicznych. W przypadku linkowania działa na tablicy zawierającej wskaźniki na obiekty dynamiczne typu Lista.

## Kod programu:

[https://github.com/Rexluu/PAMSI/tree/Final\\_versions/List\\_6/L6\\_Z1](https://github.com/Rexluu/PAMSI/tree/Final_versions/List_6/L6_Z1)

## Testy:

Dla czytelnych testów, w każdym przypadku podany rozmiar to 13 (liczba pierwsza)

### Linkowanie

Podaj ilosc elementow: 13

Dodaje:

Dana: 18 o kluczu: 5  
Dana: 41 o kluczu: 2  
Dana: 22 o kluczu: 9  
Dana: 44 o kluczu: 5  
Dana: 59 o kluczu: 7  
Dana: 32 o kluczu: 6  
Dana: 31 o kluczu: 5  
Dana: 73 o kluczu: 8  
Dana: 90 o kluczu: 12  
Dana: 28 o kluczu: 2  
Dana: 23 o kluczu: 10  
Dana: 9 o kluczu: 9  
Dana: 48 o kluczu: 9

Po dodaniu:

Miejsce 0:					
Miejsce 1:					
Miejsce 2:		41		28	
Miejsce 3:					
Miejsce 4:					
Miejsce 5:		18		44	
Miejsce 6:		32			
Miejsce 7:		59			
Miejsce 8:		73			
Miejsce 9:		22		9	
Miejsce 10:		23			
Miejsce 11:					
Miejsce 12:		90			

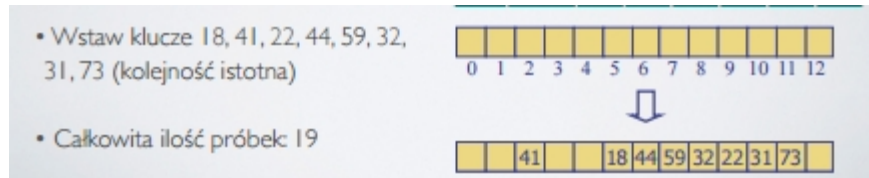
Pozycja danej 18: pos.5 #1

Po usunieciu 18:

Miejsce 0:					
Miejsce 1:					
Miejsce 2:		41		28	
Miejsce 3:					
Miejsce 4:					
Miejsce 5:		44		31	
Miejsce 6:		32			
Miejsce 7:		59			
Miejsce 8:		73			
Miejsce 9:		22		9	
Miejsce 10:		23			
Miejsce 11:					
Miejsce 12:		90			

## Próbkowanie Liniowe

Dla porównania, jak powinno to wyglądać:



Dodaje:

Dana: 18 o kluczu pierwotnym: 5  
Dana: 41 o kluczu pierwotnym: 2  
Dana: 22 o kluczu pierwotnym: 9  
Dana: 44 o kluczu pierwotnym: 5  
Dana: 59 o kluczu pierwotnym: 7  
Dana: 32 o kluczu pierwotnym: 6  
Dana: 31 o kluczu pierwotnym: 5  
Dana: 73 o kluczu pierwotnym: 8  
Dana: 33 o kluczu pierwotnym: 7  
Dana: 90 o kluczu pierwotnym: 12  
Dana: 2 o kluczu pierwotnym: 2  
Dana: 19 o kluczu pierwotnym: 6  
Dana: 33 o kluczu pierwotnym: 7  
Dana: 30 o kluczu pierwotnym: 4

Po dodaniu:

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10	11	12
90	19	41	2	33	18	44	59	32	22	31	73	33

Pozycja danej 18 : 5

Po usunięciu 18:

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10	11	12
90	19	41	2	33	0	44	59	32	22	31	73	33

## Podwójne hashowanie

Dodaje:

Dana: 18 o kluczu pierwotnym: 5  
Dana: 41 o kluczu pierwotnym: 2  
Dana: 22 o kluczu pierwotnym: 9  
Dana: 44 o kluczu pierwotnym: 5  
Dana: 59 o kluczu pierwotnym: 7  
Dana: 32 o kluczu pierwotnym: 6  
Dana: 31 o kluczu pierwotnym: 5  
Dana: 73 o kluczu pierwotnym: 8  
Dana: 76 o kluczu pierwotnym: 11  
Dana: 64 o kluczu pierwotnym: 12  
Dana: 59 o kluczu pierwotnym: 7  
Dana: 6 o kluczu pierwotnym: 6  
Dana: 61 o kluczu pierwotnym: 9  
Dana: 37 o kluczu pierwotnym: 11

Po dodaniu: (dla  $q = 2$ )

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10	11	12
76	59	41	6	61	18	32	44	59	22	31	73	64

Pozycja danej 18 : 5

Po usunięciu 18:

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10	11	12
76	59	41	6	61	0	32	44	59	22	31	73	64

W każdym z 3 przypadków, zaprezentowano funkcje dodawania, wyszukiwania oraz usuwania. W przypadku linkowania, można dodać „dowolną” ilość, jednak próbkowanie liniowe oraz podwójne hashowanie ma ograniczone miejsce. W przypadku nieznaalezienia konkretnej danej, program wyrzuci daną „-1”.

W programie została również zaimplementowana funkcja sprawdzająca czy podany rozmiar jest liczbą pierwsza:

Podaj ilosc elementow: 4

Podaj liczbe pierwsza!

## Program 2

2. Należy zaimplementować drzewo AVL przechowujące integer. Należy napisać funkcje wykonujące podstawowe operacje na drzewie (dodawanie i usuwanie elementów, wyświetlanie elementu w korzeniu drzewa, wyświetlanie wysokości drzewa),
- (a) należy wyświetlić zawartość drzewa w postaci: "korzeń: lewy syn, prawy syn, wysokość lewego poddrzewa, wysokość prawego poddrzewa, różnica wysokości",
  - (b) należy wyświetlić każdy etap rotacji węzłów podczas restrukturyzacji drzewa,

### Opis:

Należało zaimplementować drzewo AVL przechowujące integer. Drzewo AVL jest drzewem BST, gdzie dla każdego węzła różnica wysokości jego poddrzew (lewego i prawego) nie jest wyższa niż 1. W poleceniu jest wspomniane, iż **drzewo to ma przechowywać integer**, jednak **autor wykorzystał już wcześniej implementowane drzewo BTS na szablonach** i przerobił je na AVL, dlatego też szablony pojawiają się i tu.

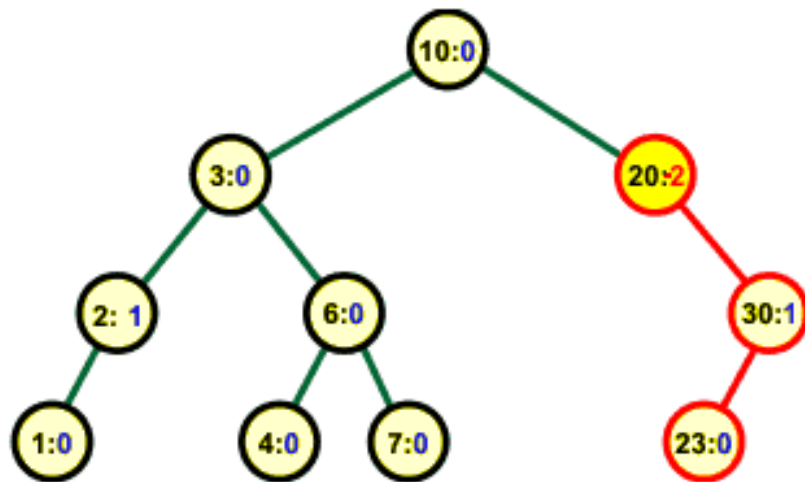
### Kod programu:

[https://github.com/Rexluu/PAMSI/tree/Final\\_versions/List\\_6/L6\\_Z2](https://github.com/Rexluu/PAMSI/tree/Final_versions/List_6/L6_Z2)

## Testy:

Do testu zostały dobrane dane tak, aby sprawdzić poprawność działania algorytmów restrukturyzacji.

Poniżej prezentacja drzewa BST, aby zilustrować, że do tych danych należy wykonać restrukturyzację drzewa BST, aby powstało drzewo AVL.



Dane: 10; 3; 20; 2; 6; 30; 1; 4; 7; 23

Preorder powyższego drzewa BST: 10; 3; 2; 1; 6; 4; 7; 20; 30; 23

Te same dane, z użyciem drzewa AVL:

Procedura powstawania drzewa wraz z restrukturyzacją:

```
Dodawana wartosc: 10
Pre-order: 10|
- - - - -
Dodawana wartosc: 3
Pre-order: 10| 3|
- - - - -
Dodawana wartosc: 20
Pre-order: 10| 3| 20|
- - - - -
Dodawana wartosc: 2
Pre-order: 10| 3| 2| 20|
- - - - -
Dodawana wartosc: 6
Pre-order: 10| 3| 2| 6| 20|
- - - - -
Dodawana wartosc: 30
Pre-order: 10| 3| 2| 6| 20| 30|
- - - - -
Dodawana wartosc: 1
Pre-order: 10| 3| 2| 1| 6| 20| 30|
- - - - -
Dodawana wartosc: 4
Pre-order: 10| 3| 2| 1| 6| 4| 20| 30|
- - - - -
Dodawana wartosc: 7
Pre-order: 10| 3| 2| 1| 6| 4| 7| 20| 30|
- - - - -
Dodawana wartosc: 23
-->Wykonuje rotacja RL
Pre-order: 10| 3| 2| 1| 6| 4| 7| 23| 20| 30|
- - - - -
```



Potwierdzenie prawidłowej restrukturyzacji, poszczególne wysokości poddrzew:

```
- - - - -
Dla wartosci: 10
Wysokosc lewego poddrzewa: 3
Wysokosc prawego poddrzewa: 2
Roznica wysokosci poddrzew: 1
- - - - -
Dla wartosci: 3
Wysokosc lewego poddrzewa: 2
Wysokosc prawego poddrzewa: 2
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 2
Wysokosc lewego poddrzewa: 1
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 1
- - - - -
Dla wartosci: 1
Wysokosc lewego poddrzewa: 0
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 6
Wysokosc lewego poddrzewa: 1
Wysokosc prawego poddrzewa: 1
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 4
Wysokosc lewego poddrzewa: 0
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 7
Wysokosc lewego poddrzewa: 0
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 23
Wysokosc lewego poddrzewa: 1
Wysokosc prawego poddrzewa: 1
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 20
Wysokosc lewego poddrzewa: 0
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 0
- - - - -
Dla wartosci: 30
Wysokosc lewego poddrzewa: 0
Wysokosc prawego poddrzewa: 0
Roznica wysokosci poddrzew: 0

Pre-order po usunieciu 1:
Pre-order: 10| 3| 2| 6| 4| 7| 23| 20| 30|
```

## Program 3

3. Należy zaimplementować graf przechowujący elementy określonego typu. Należy napisać funkcje wykonujące podstawowe operacje na grafie zgodnie z informacjami podawanymi na wykładzie.
- (a) Należy zaimplementować graf za pomocą listy sąsiedztwa zgodnie z wytycznymi na wykładzie
  - (b) Należy zaimplementować graf za pomocą macierzy sąsiedztwa zgodnie z wytycznymi na wykładzie.

### Opis:

Zadanie polegało na zaimplementowaniu grafów (skierowanych lub nieskierowanych) za pomocą listy sąsiedztwa bądź macierzy sąsiedztwa.

W programie występują zmienne dynamiczne oraz zmodyfikowana lista jednokierunkowa implementowana na wcześniejszych laboratoriach (**nie wykorzystano stl**).

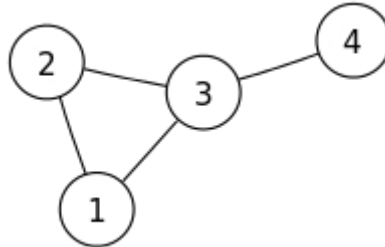
**Dodatkowo zaimplementowano od razu priorytet/wagi krawędzi. W poleceniu o nich nic nie jest napisane, dlatego też każda waga w teście będzie wynosić 0.**

### Kod programu:

[https://github.com/Rexluu/PAMSI/tree/Final\\_versions/List\\_6/L6\\_Z3](https://github.com/Rexluu/PAMSI/tree/Final_versions/List_6/L6_Z3)

## Testy:

Jako test, do obu wersji wprowadzono graf widoczny na rysunku poniżej, następnie usunięto z niego wierzchołek, którego wartość wynosi jeden oraz wypisano graf (wierzchołki, krawędzie, wagi) ponownie.



Graf oparty o macierz sąsiedztwa

Graf nieskierowany:

```
4 --- 0 ---> 3
1 --- 0 ---> 3
1 --- 0 ---> 2
3 --- 0 ---> 4
3 --- 0 ---> 1
3 --- 0 ---> 2
2 --- 0 ---> 1
2 --- 0 ---> 3
```

Po usunięciu wierzchołka 1 i jego krawędzi:

Graf nieskierowany:

```
4 --- 0 ---> 3
3 --- 0 ---> 4
3 --- 0 ---> 1
3 --- 0 ---> 2
2 --- 0 ---> 1
2 --- 0 ---> 3
```

Graf oparty o liste sąsiedztwa:

Graf nieskierowany:

```
4 --- 0 ---> 3
1 --- 0 ---> 2
1 --- 0 ---> 3
3 --- 0 ---> 2
3 --- 0 ---> 1
3 --- 0 ---> 4
2 --- 0 ---> 1
2 --- 0 ---> 3
```

Po usunięciu wierzchołka 1 i jego krawędzi:

Graf nieskierowany:

```
4 --- 0 ---> 3
3 --- 0 ---> 2
3 --- 0 ---> 1
3 --- 0 ---> 4
2 --- 0 ---> 1
2 --- 0 ---> 3
```

Dodatkowo za pomocą programu valgrind, sprawdzono czy występują wycieki pamięci

```
==7142==
==7142== HEAP SUMMARY:
==7142==    in use at exit: 0 bytes in 0 blocks
==7142==   total heap usage: 16 allocs, 16 frees, 424 bytes allocated
==7142==
==7142== All heap blocks were freed -- no leaks are possible
==7142==
==7142== For counts of detected and suppressed errors, rerun with: -v
==7142== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```