
	Uniwersytet Technologiczno-Przyrodniczy Im. J. J. Śniadeckich w Bydgoszczy Wydział Telekomunikacji, Informatyki i Elektrotechniki <b>Zakład Techniki Cyfrowej</b>		
<b>Przedmiot</b>	Algorytmy i Struktury Danych		
<b>Prowadzący</b>			
<b>Temat</b>	<i>Struktury danych w bibliotece STL</i>		
<b>Student</b>			
<b>Nr ćw.</b>	12	<b>Data wykonania</b>	
<b>Ocena</b>		<b>Data oddania spr.</b>	

## 1. Cel ćwiczenia

Celem ćwiczenia jest poznanie struktur danych, które są zaimplementowane w standardowej bibliotece STL oraz ich użycie w przykładowych zadaniach.

## 2. Informacje podstawowe

### 2.1. Kontenery

Są to po prostu struktury, które przechowują dane w sposób uporządkowany. Należy jednak pamiętać, że dane muszą mieć jeden typ. W zależności od wybranego kontenera można uzyskać lepsze lub gorsze rezultaty w zakresie wydajności. W bibliotece zaimplementowano:

- listę,
- wektor,
- mapę,
- zbiór,
- stos,
- kolejki.

### 2.2. Stos

Był on tematem poprzednich zajęć, więc jego działanie jest doskonale znane. W bibliotece STL ma następujące operacje:

- *push* – odłożenie elementu na górę stosu,
- *pop* – zdjęcie górnego elementu ze stosu,
- *empty* – informacja czy stos jest pusty,
- *size* – informacja o wielkości stosu, aktualna liczba jego elementów,
- *top* – pobranie wartości najwyżej położonego elementu.

Implementacja i działanie stosu zostały przedstawione na poniższym przykładzie:

```
stack<string> stos;
cout << "wstawiam na stos wyrazy: ala, ma, kota.\n";
stos.push("ala");
stos.push("ma");
stos.push("kota");
cout << "najwyższy element stosu to: " << stos.top() << "\n";
stos.top() = "kotka";
cout << "najwyższy element jest zmodyfikowany i ma wartosc: " << stos.top() << "\n";
cout << "ilosc elementow to : " << stos.size() << "\n";
cout << "zdejmuje elementy ze stosu\n";
stos.pop();
stos.pop();
stos.pop();
if (stos.empty()) {
    cout << "stos jest pusty\n";
}
```

## 2.2. Kolejka

Była ona tematem jednych z poprzednich zajęć, więc jej działanie jest doskonale znane. W bibliotece STL ma następujące operacje:

- *push* – dodanie elementu na koniec kolejki,
- *pop* – usunięcie początkowego elementu kolejki,
- *empty* – informacja czy kolejka jest pusta,
- *size* – informacja o wielkości kolejki, aktualna liczba jej elementów,
- *front* – pobranie wartości pierwszego elementu kolejki,
- *back* – pobranie wartości ostatniego elementu kolejki.

Implementacja i działanie kolejki zostały przedstawione na poniższym przykładzie:

```
queue<string> kolejka;
cout << "wstawiam do kolejki wyrazy: ala, ma, kota.\n";
kolejka.push("ala");
kolejka.push("ma");
kolejka.push("kota");
cout << "pierwszy element kolejki to: " << kolejka.front() << "\n";
cout << "ostatni element kolejki to: " << kolejka.back() << "\n";
kolejka.front() = "Ala";
kolejka.back() = "kotka";
cout << "pierwszy el. jest zmodyfikowany, ma wartosc: " << kolejka.front() << "\n";
cout << "ostatni el. jest zmodyfikowany, ma wartosc: " << kolejka.back() << "\n";
cout << "ilosc elementow to : " << kolejka.size() << "\n";
cout << "zdejmuje elementy z kolejki\n";
kolejka.pop();
kolejka.pop();
kolejka.pop();
if (kolejka.empty()) {
    cout << "kolejka jest pusta\n";
}
```

### 2.3. Wektor

Jest to struktura, która swoją budową przypomina tablicę. Nie trzeba jednak znać wielkości tej struktury w momencie rozpoczęcia działania. W bibliotece STL ma następujące operacje:

- *push\_back* – dodanie elementu na koniec tablicy,
- *pop\_back* – usunięcie elementu z końca tablicy,
- *insert* – dodanie elementu we wskazane miejsce w tablicy,
- *size* – informacja o wielkości tablicy, aktualna liczba jej elementów,
- *front*, *back* – pobranie wartości pierwszego i ostatniego elementu tablicy.

Implementacja i działanie kolejki zostały przedstawione na poniższym przykładzie:

```
vector<string> wektor;
cout << "wstawiam do kolejki wyrazy: ala, ma, kota.\n";
wektor.push_back("ala");
wektor.push_back("ma");
wektor.push_back("kota");
cout << "wstawiam do kolejki wyraz: malego.\n";
wektor.insert(wektor.begin() + 2, "malego");
cout << "wypisanie wektora:\n";
for (int i = 0; i < wektor.size(); i++)
{
    cout << wektor[i] << " ";
}
cout << "\n";
wektor.front() = "Ala";
wektor.back() = "kotka";
cout << "pierwszy el. jest zmodyfikowany, ma wartosc: " << wektor.front() << "\n";
cout << "ostatni el. jest zmodyfikowany, ma wartosc: " << wektor.back() << "\n";
cout << "ilosc elementow to : " << wektor.size() << "\n";
cout << "zdejmuje elementy z kolejki\n";
wektor.pop_back();
wektor.pop_back();
wektor.pop_back();
wektor.pop_back();
if (wektor.empty()) {
    cout << "wektor jest pusty\n";
}
```

### 3. Przebieg ćwiczenia

#### 3.1. Zadanie 1.

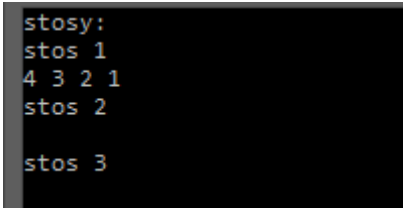
Napisać program, który będzie wyświetlał menu i w zależności od podanych od użytkownika danych utworzy stos, kolejkę lub wektor. Dla każdej struktury umożliwić:

- dodawanie i usuwanie elementów,
- pobranie wartości skrajnych elementów,
- pobranie n-tego elementu struktury,
- sprawdzenie czy element należy do struktury,
- pobranie wielkości struktury,
- sprawdzenie, czy nie jest pusta.

Skopiować treść rozwiązania, aby umieścić je w sprawozdaniu.

#### 3.2. Zadanie 2.

Napisać program, który za pomocą trzech stosów umożliwi rozwiązywanie przez użytkownika problemu wież Hanoi dla wczytanej liczby od użytkownika. Przykładowy interfejs przedstawiono poniżej:



```
stosy:
stos 1
4 3 2 1
stos 2
stos 3
```

Skopiować treść rozwiązania, aby umieścić je w sprawozdaniu.

#### 3.3. Zadanie 3.

Napisać program, który za stosu i kolejki zmieni zapis z nawiasami na zapis ONP. Algorytm zamiany przedstawić również w postaci schematu blokowego.

Skopiować treść rozwiązania, aby umieścić je w sprawozdaniu.

### 4. Sprawozdanie

Sprawozdanie z laboratorium powinno zawierać:

- wypełnioną tabelę z początku instrukcji
- schematy blokowe algorytmów,
- kody programów będących rozwiązaniami wszystkich zadań wraz z komentarzami,
- wnioski.