STL, czyli o co tyle hałasu

W świecie programowania C++, hasło STL pojawia się nieustannie i zawsze jest o nim głośno... często początkujące osoby, które nie znają STL-a pytają się co to jest i czemu go tak wszyscy zachwalają. Bardzo często spotykamy się z następującą odpowiedzią: STL jest to zbiór kontenerów, iteratorów i algorytmów - naprawdę świetna sprawa, nie wiem jak bez tego można pisać programy... - ale co to tak naprawdę znaczy? Ostatni termin tj. algorytmy jest zrozumiały, jednak algorytmy w STL-u nie powalają z nóg... raczej można powiedzieć, że biblioteka ta jest uboga, a co gorsza nie ma tam nic co mogłoby się przydać bezpośrednio do gier (np. algorytmy wykrywania kolizji itp - tego w STL-u nie znajdziesz).

Definicja kontenera

Kontener jest to struktura danych, która służy do przechowywania danych w zorganizowany sposób. Wszystkie elementy kontenera **muszą być** takiego samego typu. Każdy kontener umożliwia nam wykonanie takich operacji jak:

- uzyskanie dostępu do danych w kontenerze;
- możliwość dodawania elementów do kontenera;
- możliwość usuwania elementów z kontenera.

Niektóre z nich dają nam również możliwość wyszukiwania elementu w kontenerze. Kontenery w zależności od przyjętej organizacji danych różnią się **szybkością** wykonywania poszczególnych operacji.

Dokumentacja kontenera vector

Kompletna dokumentacja kontenera C++ vector w - http://www.cplusplus.com/reference/stl/vector/

Co to jest vector?

Vector jest to tak zwany **kontener na dane** (pojemnik), inaczej dynamiczna tablica. W owej tablicy mamy dostęp do każdego jej elementu oraz możemy w każdym momencie zwiększać jej wielkość.

Jakimi funkcjami posługujemy się używając vectora?

- **push_back();** - dodaje do końca tablicy nowy element podany w nawiasie
- **insert();** - dodaje element do dynamicznej tablicy w podanym miejscu
- **begin();** - wskazuje pierwszy element dynamicznej tablicy
- **end();** - wskazuje na koniec dynamicznej tablicy
- **size();** - zwraca ilość elementów tablicy

Przykładowe kontenery:

- tablica
- tablica asocjacyjna
- lista
    - lista jednokierunkowa
    - lista dwukierunkowa
- drzewo
    - drzewo binarne

Co to jest złożoność obliczeniowa?

Szybkość nazywana jest przez programistów **złożonością obliczeniową** i oznaczana jest przez dużą literę **O**. Złożoność obliczeniowa jest to ilość podstawowych operacji jakie trzeba wykonać, aby został zrealizowany dany algorytm, posiadając określoną ilość danych wejściowych. Przykłady:

- O(K*1) - stała złożoność - algorytm wykonuje się w stałej ilości operacji, niezależnie od ilości danych. (algorytm: **najszybszy**)
- O(log(K*n)) - złożoność logarytmiczna - algorytm wykonuje logarytmiczną ilość operacji w stosunku do ilości danych wejściowych; **n** jest to ilość danych wejściowych. Podstawa logarytmu zazwyczaj wynosi 2, jednak czasami może być większa (np. w B-drzewach - nie będą one jednak tematem tego kursu). (algorytm: **bardzo szybki**)
- O(K*n) - złożoność liniowa - algorytm wykonuje wprost proporcjonalną ilość operacji do ilości danych wejściowych. (algorytm: **szybki**)
- O(K*n^X) - złożoność wielomianowa - ilość operacji algorytmu jest wprost proporcjonalna do ilości danych wejściowych podniesionej do potęgi **X**, gdzie **X** jest stałą większą lub równą 2. (algorytm: **wolny**)
- O(K*X^n) - złożoność wykładnicza - ilość operacji algorytmu jest wprost proporcjonalna do stałej X większej lub równej 2, podniesionej do potęgi równej ilości danych wejściowych. (algorytm: **bardzo wolny**)

K - jest to stała, która może mieć wartość zarówno 1 jak i 1000000. Zazwyczaj stała ta jest tak mała, że jest ona pomijalna przy podawaniu złożoności obliczeniowej algorytmów. Najczęściej wynosi ona 1 lub 2.

złożoność obliczeniowa	**ilość operacji**	**ilość danych wejściowych**
O(1)	1	100 000
O(log(n))	17	100 000
O(n)	100 000	100 000
O(n^2)	10 000 000 000	100 000
O(2^n)	1 267 650 600 228 230 000 000 000 000 000	100
O(n^n)	88 817 841 970 012 500 000 000 000 000 000 000	25
Procesor 3.0GHz/sek	3 000 000 000	

Kontenery dostępne w STL-u

W STL-u znajduje się kilka kontenerów do zarządzania danymi. Każdy z nich posiada inną złożoność obliczeniową dla poniższych operacji:

- odczyt danych;
- dodawanie danych;
- usuwanie danych;
- szukanie danych.

Jak już wcześniej wspomniałem wyszukiwanie danych nie jest dostępne w niektórych kontenerach, jednak jest to zabieg celowy o czym przekonasz się później. Poniżej przedstawiam listę kontenerów jakie znajdziesz w STL-u:

- lista (**list**)
- tablica (**vector**)
- tablica podwójnie kończona (**deque**)
- tablica bitowa (**bitset**)
- drzewo poszukiwań (**set**)
- wielokrotne drzewo poszukiwań (**multiset**)
- mapa poszukiwań (**map**)
- wielokrotna mapa poszukiwań (**multimap**)

Oprócz kontenerów w STL-u istnieją również adaptery:

- stos (**stack**)
- kolejka (**queue**)
- kolejka priorytetowa (**priority_queue**)

Opisy poszczególnych kontenerów (i adapterów), złożoności obliczeniowe operacji, przykłady itp. będą przedstawiane stopniowo w kolejnych rozdziałach.

Co to są adaptery?

Adapter jest jednym z wzorców projektowych. Zadaniem adaptera jest przekształcanie interfejsów różnych klas w taki, który jest oczekiwany przez użytkownika. Innymi słowy adapter daje nam metody, za pomocą których możemy np. pobierać i zapisywać dane w określony sposób, ale sposób organizacji danych w adapterze jest nieznany. Dzięki adapterowi możemy zmienić klasę zarządzającą danymi nie zmieniając działania całej aplikacji.

Co to są iteratory?

Iterator jest to obiekt pozwalający na sekwencyjny dostęp do wszystkich danych, znajdujących się w konkretnym kontenerze. Dzięki niemu możemy w łatwy sposób poruszać się po kontenerze, usuwać wybrane elementy lub napisać wyszukiwanie o złożoności obliczeniowej liniowej, jeśli dany kontener nie posiada wyszukiwania.

Napisz program, który w dynamicznej tablicy będzie gromadził znaki wprowadzone z klawiatury.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
 vector <char> znaki;
 char c;
 int i;
 cout << "Podawaj znaki, a ja bede je zapamietywac ..." << endl;
 cout << "Q - koniec" << endl;
 do
 {
    cin >> c;
    znaki.push_back( c);

 } while( c != 'q');
 cout << "Oto znaki:" << endl;

 for( i = 0; i < znaki.size(); i ++)
 {
    cout << znaki[ i] << endl;
 }
 stop();
 return 0;
}
```

Napisz program, który będzie gromadził w kontenerze wpisywane z klawiatury imiona.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
 vector <string> imiona;
 string s;
 int i;

 cout << "Podawaj imiona, a ja bede je zapamietywac ..." << endl;
 cout << "Q - koniec" << endl;
 while( true)          //wieczna pętla
 {
    cin >> s;
    if( s == "q" || s == "Q")
       break;
    imiona.push_back( s);

 }
 cout << "Oto imiona:" << endl;
 for( i = 0; i < imiona.size(); i ++)
 {
    cout << imiona[ i] << endl;
 }
...
```

Napisz program, który będzie losować i umieszczać w kontenerze liczby z zakresu 1000 liczb. Po załadowaniu kontenera wyświetl informacyjnie jakiś podzbiór tych liczb.

```
#include <iostream>
#include <vector>
using namespace std;
void stop( void);
//-------------------------------------
int main()
{
 vector <int> liczby;
 int i, ile = 1000;

 cout << "Losowanie " << ile << " liczb." << endl;
 for( i = 0; i < ile; i ++)
 {
    liczby.push_back( rand());
 }
 cout << "Zakonczono zapelnianie kontenera." << endl;
 for( i = 100; i < 110; i ++)
 {
   cout << "Liczba nr " << i << " = ";
   cout.width( 10);
   cout << liczby[ i] << endl;
 }
 stop();
 return 0;
}
...
```

Tak zmodyfikuj poprzedni program , by kontener zawierał posortowany zbiór liczb. To znaczy kolejne liczby nie wkładaj na koniec kontenera, ale od razu na ich właściwe miejsca wyznaczone ich wielkością.

```
...
vector <int> liczby;
int a, i, j, pozycja, ile = 1000;

cout << "Losowanie " << ile << " liczb." << endl;
for( i = 0; i < ile; i ++)
{
   a = rand();          //kolejny obiekt do kontenera...
   pozycja = 0;         //... i wstępnie sugerowana pozycja w kontenerze
   for( j = 0; j < liczby.size(); j ++)
   {
      if( a > liczby[ j])
      {
         pozycja ++;  //szukamy miejsca w szeregu dla obiektu a
      }
   }
   liczby.insert( liczby.begin() + pozycja, a);
}
cout << "Zakonczono zapelnianie kontenera." << endl;
for( i = 100; i < 110; i ++)
{
   cout << "Liczba nr " << i << " = ";
   cout.width( 10);
   cout << liczby[ i] << endl;
}
...
```

Posortuj bezładnie zapełniony kontener

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
...
vector <int> liczby;
int i, ile = 1000;
cout << "Losowanie " << ile << " liczb." << endl;
for( i = 0; i < ile; i ++)
{
   liczby.push_back( rand());
}
cout << "Zakonczono zapelnianie kontenera." << endl;
sort( liczby.begin(), liczby.end());    //sortowanie kontenera
for( i = 800; i < 810; i ++)
{
   cout << "Liczba nr " << i << " = ";
   cout.width( 10);
   cout << liczby[ i] << endl;
}
```

Zadanie 1 (2p)

Stwórz kontener do przechowywania liczb typu float i umieść w nim 10 losowych liczb.

Zadanie 2 (4p)

Korzystając z stworzonej na poprzednich ćwiczeniach klasie pojazd napisz kontener dla tej klasy. Stwórz dwa obiekty klasy pojazd i dodaj je do kontenera. Dodaj możliwość dodawania nowych pojazdów do kontenera, wypisywania rozmiaru kontenera i wypisywania informacji o danym pojeździe zawartym w kontenerze.

Zadanie 3 (2p)

Napisz program, który będzie losować i umieszczać w kontenerze zadaną ilość liczb, ale w sposób uporządkowany od najmniejszej do największej. Sporządź kontrolny wydruk zawartości kontenera. Obowiązkowy interfejs programu z użytkownikiem.