ANGELIKA JEZIORSKA, JAGODA CHMIELEWSKA

KONTENERY STL

RODZAJE KONTENERÓW

- Sekwencyjne
  - vector (tablica dynamiczna)
  - dequeue (tablica dynamiczna dwukierunkowa)
  - list (lista dwukierunkowa)

- array (C++11) – tablica o stałym rozmiarze, przekazanym jako drugi argument szablonu
- forward_list (C++11) – lista jednokierunkowa

- tablica (z C) i string - **nie są kontenerami STL**, ale można je przetwarzać za pomocą algorytmów STL
- adaptery kontenerów
  - stack (kolejka LIFO), queue (kolejka FIFO), priority_queue

RODZAJE KONTENERÓW

- Asocjacyjne
  - set (drzewo binarne)
  - multiset
  - map, multimap
- Asocjacyjne nieuporządkowane (C++11)
- (zaimplementowane za pomoca tablicy mieszającej zamiast drzewa binarnego, w starszych implementacjach znane jako hash_*)
  - unordered_set, unordered_multiset,
  - unordered_map, unordered_multimap

VECTOR

- implementacja jako tablica dynamiczna
- szybkie push_back()
- szybki operator[]
- wolne insert()

DEQUE

- implementacja jako tablica dynamiczna
- szybkie push_back() (vector może być szybszy)
- szybkie push_front() (nie ma w vectorze)
- szybki operator[]
- wolny insert()

LIST

- implementacja jako lista **dwukierunkowa**
- szybkie push_back()
- szybkie push_front()
- szybki insert()
- brak operator[]

FORWARD_LIST

- implementacja jako lista **jednokierunkowa**
- szybkie push_back() (vector może być szybszy)
- szybkie insert_after()
- brak operator[]
- brak push_back()

ITERATORY

- Iteratory działają jak wskaźniki, ale działają dla **wszystkich kontenerów**:
  - `* ->`
  - `++  --`
  - `==  !=`
  - `=`
  - (vector, deque, array, string: -, <, >, +(int))
  - container.begin() (zwróć iterator do pierwszego elementu)
  - container.end() (do ostatniego)

SET

- implementacja jako drzewo binarne
- sortowanie przy wstawianiu (porównywanie przez domyślny operator<())
- odpowiednik zbiorów matematycznych
- metoda find()
- szybkie insert()
- brak: push_back(), push_front(), operator[]

MULTISET

- możliwe powtórzenia o tych samych wartościach
- uporządkowanie wewnątrz grupy elementów tej samej wartości jest niezdefiniowane

MAP

- zazwyczaj implementowane jako drzewa czerwono-czarne
- set dla par: klucz, wartość – użyj make_pair()
- tylko dla kontenera map: operator[key]()
  - indeksowanie wartością (tablica asocjacyjna)
  - elementy są sortowane na podstawie funkcji porównującej Compare

MULTIMAP

- multiset dla par - pozwala na przechowywanie posortowanej listy par postaci klucz-wartość
- możliwe jest przechowywanie wielu elementów o tym samym kluczu
- set dla par: klucz, wartość – użyj make_pair()

ADAPTERY KONTENERÓW

- stack (kolejka LIFO)
- queue (kolejka FIFO)
- priority_queue (kolejka priorytetowa)

STACK

- header <stack>
- prosta koleja LIFO zaimplementowana za pomocą deque
  - może być oparta o kontener mający: back(), push_back(), pop_back()
  - wystarczy podać 2gi argument szablonu, np.: stack<int, vector<int>> po prostu przekierowuje swoje metody do deque aby uzyskać kolejkę LIFO
- push()
- pop()
- top() (pobierz wartość elementu ze szczytu stosu, ale nie usuwaj go)
- size() (zwróć rozmiar stosu)
- empty() (czy stos jest pusty?)
- operatory : ==, !=, <, >, <=, >=   (== zwraca 1 gdy rozmiary i elementy identyczne)
- **innych operatorów nie ma**

QUEUE

- header `<queue>`
- prosta kolejka FIFO zaimplementowana za pomocą `deque`
  - może być oparta o kontener mający : `front(), back(), push_back(), pop_front()`
  - wystarczy podać 2gi argument szablonu, np.: `queue<int, list<int>>` po prostu przekierowuje swoje metody do `deque` aby uzyskać kolejkę FIFO

- `push()`
- `pop()`
- `back()`(zwróć wartość ostatnio wstawionego elementu, ale go nie usuwaj)
- `front()`(zwróć wartość elementu z przodu kolejni, ale go nie usuwaj)
- `size()`(zwróć rozmiar kolejki)
- `empty()`(czy kolejka jest pusta?)
- operatory: `==, !=, <, >, <=, >=`
- **innych operatorów i metod nie ma**

PRIORITY_QUEUE

- header <queue>
- prosta kolejka priorytetowa zaimplementowana za pomocą vector
  - może być oparta o kontener mający: front(), push_back(), pop_front(), operator[]
  - wystarczy podać 2gi argument szablonu, np.: priority_queue<int, deque<int>>,3ci argument to kryterium sortowania (domyślnie: operator<)
- priority_queue<int, vector<int>, greater<int>>
- przekierowuje swoje metody do metod kontenera vector

- push()
- pop()
- top() (zwróć wartość elementu o największym priorytecie, ale go nie usuwaj z kolejki)
- size() (liczba elementów)
- empty() (czy kolejka pusta)
- brak innych metod lub operatorów (w tym operatora ==, etc.)

PRZYKŁADY UŻYCIA KONTENERÓW DO WŁASNEJ ANALIZY:

VECTOR:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <iterator>
using namespace std;

int main()
{
    // utworz pusty wektor na lancuchy
    vector<string> sentence;

    // zarezerwuj pamiec na 5 elementow, aby uniknac realokacji
    sentence.reserve(5);

    // dolacz kilka elementow
    sentence.push_back("Witaj,");
    sentence.push_back("jak");
    sentence.push_back("sie");
    sentence.push_back("masz");
    sentence.push_back("?");

    // wypisz elementy rozdzielone spacjami
    copy (sentence.begin(), sentence.end(),
          ostream_iterator<string>(cout," "));
    cout << endl;

    // wypisz 'dane techniczne'
    cout << "  max_size(): " << sentence.max_size() << endl;
    cout << "  size():     " << sentence.size()     << endl;
    cout << "  capacity(): " << sentence.capacity() << endl;

    // zamien drugi element z czwartym
    swap (sentence[1], sentence[3]);

    // przed elementem "?" wstaw element "zawsze"
    sentence.insert (find(sentence.begin(),sentence.end(),"?"),
                     "zawsze");

    // przypisz "!" do ostatniego elementu
    sentence.back() = "!";

    // wypisz elementy rozdzielone spacjami
    copy (sentence.begin(), sentence.end(),
          ostream_iterator<string>(cout," "));
    cout << endl;

    // ponownie wypisz 'dane techniczne'
    cout << "  max_size(): " << sentence.max_size() << endl;
    cout << "  size():     " << sentence.size()     << endl;
    cout << "  capacity(): " << sentence.capacity() << endl;
}
```

PRZYKŁADY UŻYCIA KONTENERÓW DO WŁASNEJ ANALIZY:

DEQUE

```
#include <iostream>
#include <deque>
#include <string>
#include <algorithm>
#include <iterator>

using namespace std;

int main()
{
    // utworz pusta kolejke deque na lancuchy
    deque<string> coll;

    // wstaw kilka elementow
    coll.assign (3, string("lancuch"));
    coll.push_back ("ostatni lancuch");
    coll.push_front ("pierwszy lancuch");

    // wypisz elementy w oddzielnych wierszach
    copy (coll.begin(), coll.end(),
          ostream_iterator<string>(cout,"\n"));
    cout << endl;

    // usun pierwszy i ostatni element
    coll.pop_front();
    coll.pop_back();

    // wstaw lancuch "inny" do kazdego elementu oprocz pierwszego
    for (unsigned i=1; i<coll.size(); ++i) {
        coll[i] = "inny " + coll[i];
    }

    // zmien rozmiar na 4 elementy
    coll.resize (4, "lancuch zmiany rozmiaru");

    // wypisz elementy w oddzielnych wierszach
    copy (coll.begin(), coll.end(),
          ostream_iterator<string>(cout,"\n"));
}
```

LIST

```
 // utworz dwie puste listy
    list<int> list1, list2;

    // wypelnij obydwie listy elementami
    for (int i=0; i<6; ++i) {
        list1.push_back(i);
        list2.push_front(i);
```

PRZYKŁADY UŻYCIA KONTENERÓW DO WŁASNEJ ANALIZY:

SET:

```
#include <iostream>
#include <set>
#include <iterator>
using namespace std;

int main()
{
    /* typ kolekcji:
     * - brak powtorzen
     * - elementy sa wartosciami calkowitymi
     * - porzadek malejacy
     */
    typedef set<int,greater<int> > IntSet;

    IntSet coll1;        // pusty zbior

    // wstaw elementy w przypadkowej kolejnosci
    coll1.insert(4);
    coll1.insert(3);
    coll1.insert(5);
    coll1.insert(1);
```

MULTISET:

```
#include <iostream>
#include <set>
#include <iterator>
using namespace std;

int main()
{
    /* typ kolekcji:
     * - powtorzenia dozwolone
     * - elementy sa wartosciami calkowitymi
     * - porzadek malejacy
     */
    typedef multiset<int,greater<int> > IntSet;

    IntSet coll1;        // pusty wielozbior

    // wstaw elementy w przypadkowej kolejnosci
    coll1.insert(4);
    coll1.insert(3);
    coll1.insert(1);
    coll1.insert(2);
```

PRZYKŁADY UŻYCIA KONTENERÓW DO WŁASNEJ ANALIZY:

MAP:

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
    /* utworz mape / tablice asocjacyjna
     * - klucze sa typu string
     * - wartosci sa typu float
     */
    typedef map<string,float> StringFloatMap;

    StringFloatMap stocks;      // utworz pusty kontener

    // wstaw kilka elementow
    stocks["Agora"] = 44.50;
    stocks["COMPLAND"] = 86.00;
    stocks["Zywiec"] = 348.00;
    stocks["KGHM"] = 13.30;
    stocks["Okocim"] = 13.15;
```

MULTIMAP:

```
#include <iostream>
#include <map>
#include <string>
#include <iomanip>
using namespace std;

int main()
{
    // zdefiniuj typ multimap jako slownik typu string-string
    typedef multimap<string,string> StrStrMMap;

    // utworz pusty slownik
    StrStrMMap dict;

    // wstaw kilka elementow w przypadkowej kolejnosci
    dict.insert(make_pair("day","dzien"));
    dict.insert(make_pair("strange","obcy"));
    dict.insert(make_pair(„car","samochod"));
```

PRZYKŁADY UŻYCIA ADAPTERÓW KONTENERÓW DO WŁASNEJ ANALIZY:

QUEQUE:

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
    queue<string> q;

    // do kolejki wstaw trzy elementy
    q.push("Tu ");
    q.push("sa ");
    q.push("wiecej niz ");

    // z kolejki odczytaj i wypisz dwa elementy
    cout << q.front();
    q.pop();
    cout << q.front();
    q.pop();

    // wstaw dwa nowe elementy
    q.push("cztery ");
    q.push("slowa!");

    // opusc jeden element
    q.pop();

    // odczytaj i wypisz dwa elementy
    cout << q.front();
    q.pop();
    cout << q.front() << endl;
    q.pop();

    // wypisz liczbe elementow w kolejce
    cout << "liczba elementow w kolejce: " << q.size()
         << endl;
}
```

STACK:

```
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    stack<int> st;

    // poloz trzy elementy na stos
    st.push(1);
    st.push(2);
    st.push(3);
```