Programowanie II

Lista zadań nr 12

Biblioteka standardowa STL

Pomimo, że język C++ ma już „swoje lata” to wciąż jest rozwijany. Dotyczy to szczególnie jego biblioteki standardowej określanej skrótem STL (Standard Template Library). Na STL składają się szablony:

- Kontenerów
  - String
  - Vector
  - lSet
  - Map
  - Stack
- Iteratorów
- Algorytmów

Biblioteka STL reprezentuje koncepcję tzw. programowania generycznego, które polega na takiej implementacji rozwiązania problemu, aby było ono możliwie niezależne od szczegółów technicznych w praktyce od zastosowanych algorytmów oraz typów danych. W STL algorytmy są niezależne od struktur danych, na których mają operować (ten sam algorytm może wykonywać operacja zarówno na wektorze jak i na liście). Rozdziału pomiędzy algorytmami oraz strukturami danych dokonano przez zastosowanie iteratorów, które stanowią swego rodzaju łącznik, pomiędzy danymi, a algorytmem na nich operującym. Struktury danych mogą być budowane zarówno z typów prostych jak i dowolnych innych klas. Również działanie samych algorytmów, może być modyfikowane przy wykorzystaniu funktorów (np. funkcji porównującej w przypadku algorytmy sortowania).

Lista funkcji zawartych w bibliotece <algorithm>

Operacje niemodyfikujące danych

- for_each
- count
- count_if
- equal
- mismatch
- find
- find_if
- find_end
- find_first_of
- adjacent_find
- search
- search_n

Operacje modyfikujące dane

- copy
- copy_backward
- fill
- fill_n
- generate
- generate_n
- transform
- remove
- remove_if
- remove_copy
- remove_copy_if
- replace
- replace_if
- replace_copy
- replace_copy_if

Operacje zmieniające kolejność

- partition
- stable_partition
- random_shuffle
- reverse
- reverse_copy
- rotate
- rotate_copy
- unique
- unique_copy
- swap
- swap_ranges
- iter_swap

Operacje sortujące

- sort
- partial_sort
- partial_sort_copy
- stable_sort
- nth_element

Operacje na posortowanych danych

- lower_bound

- upper_bound
- binary_search
- equal_range

Operacje na zbiorze

- merge
- inplace_merge

- includes
- set_difference
- set_intersection
- set_symmetric_difference
- set_union

Operacje min max

- max
- max_element
- min
- min_element
- lexicographical_compare
- next_permutation
- prev_permutation

Przykład 1 – kontener listy dwukierunkowej

```
/*
Kontener list implementuje listę dwukierunkową, jest zbliżona funkcjonalnie do
wektorów, tylko w przypadku tamtych dostęp może być realizowany zarówno
przez podanie indeksu jak iteratory. W przypadku list korzystać można jedynie z iteratorów.
*/

#include "pch.h"
#include <iostream>
#include <list>
#include <cstddef>

using namespace std;

void wyswietl(list<int> wyswietlana)
{
    unsigned rozmiar = wyswietlana.size();
    cout << "Lista sklada sie z "<<rozmiar << " elementow."<<endl;
    for (list<int>::iterator wskaznikListy = wyswietlana.begin(); wskaznikListy != wyswietlana.end(); wskaznikListy++)
        cout<<*wskaznikListy<<", ";
    cout << endl;
}
int main()
{
    list<int> lista;
    int dodawanyElement;
    cout << "Podaj kolejne elementy listy, podaj zero aby zakonczyc:\n";
    while (cin >> dodawanyElement && dodawanyElement  != 0)
        lista.push_back(dodawanyElement);
    wyswietl(lista);
    // usuniecie liczb większych niż 10
    for (list<int>::iterator wskaznikListy = lista.begin(); wskaznikListy != lista.end(); )
        if (*wskaznikListy > 10)
            wskaznikListy = lista.erase(wskaznikListy);
        else
            wskaznikListy++;
    wyswietl(lista);
    //wyswietlamy liste od tylu
    for (list<int>::reverse_iterator wskaznikListy = lista.rbegin(); wskaznikListy != lista.rend(); wskaznikListy++)
        cout << *wskaznikListy << ", ";

    return 0;
}
```

```
Podaj kolejne elementy listy, podaj zero aby zakonczyc:
1
2
3
4
67
9
0
Lista sklada sie z 6 elementow.
1, 2, 3, 4, 67, 9,
Lista sklada sie z 5 elementow.
1, 2, 3, 4, 9,
9, 4, 3, 2, 1,
```

Przykład 2 – kontener zbioru

```cpp
/*
Zbiory są strukturą opartą na drzewach, elementy w niej
przechowywane posortowane są rosnąco (taki zbiór uporządkowany)
Drzewiata struktura ułatwia szybkie wyszukiwanie, ale skutkuje
mniejszą wydajnością aktualizacji
*/

#include "pch.h"
#include <iostream>
#include <string>
#include <set>

using namespace std;

int main()
{
    set<string> zbior;
    zbior.insert("C++");
    zbior.insert("Pascal");
    zbior.insert("Java");
    zbior.insert("PHP");

    set<string>::iterator wynikSzukania, wskaznikZbioru;


    // wyświetlenie zawartości
    for (wskaznikZbioru = zbior.begin(); wskaznikZbioru != zbior.end(); ++wskaznikZbioru)
        cout << *wskaznikZbioru << '\n';

    return 0;
}
```

```
C++
Java
PHP
Pascal
```

Przykład 3 – kontener mapy (odwzorowania)

```cpp
/*
Mapa to posortowany kontener asocjacyjny (pary "identyfikator - wartość")
Elementy są ułożone w koeljności rosnącej w oparciu o identyfikatory.


*/

#include "pch.h"
#include<iostream>
#include<map>
#include <string>
#include <iterator>
using namespace std;

void wyswietlMape(map<string, int> wyswietlana, string komentarz="")
{
    map<string,int>::iterator wskaznikMapy;
    cout << "-------------------------------------------------------\n";
    cout << komentarz << endl;
    for (wskaznikMapy = wyswietlana.begin(); wskaznikMapy != wyswietlana.end(); ++wskaznikMapy)
        cout << wskaznikMapy->first <<","<<wskaznikMapy->second<< '\n';
    cout << "-------------------------------------------------------\n";
}
int main()
{
    map<string, int> mapaDni;
    //dodajemy parę do mapy - przy użyciu metody make_par
    mapaDni.insert(make_pair("poniedzialek", 1));
    mapaDni.insert(make_pair("wtorek", 2));
    wyswietlMape(mapaDni,"Zawartosc mapy na poczatku");

    /*
    dodajemy kolejny element odwołyjąc sie po indeksie - może być napisem
    warto zwrócić uwagę na kolejność - środa pojawia się przed wtorkiem
    */
    mapaDni["sroda"] = 4;
    wyswietlMape(mapaDni, "Zawartosc mapy po dodaniu srody");
    /*
    jeśli odwołamy się od identyfikatora, który już istnieje
    to para ta zostanie zaktualizowana
    */
    mapaDni["sroda"] = 3;
    wyswietlMape(mapaDni, "Zawartosc mapy po zmianie srody");
    /*
    istnieje możliwość zweryfikowania poprawności wstawiania
    */
    if (mapaDni.insert(make_pair("sroda", 3)).second == false)
        cout << "Sroda juz jest w mapie" <<endl;
    wyswietlMape(mapaDni, "Zawartosc mapy po probie aktualizacji srody");
    if (mapaDni.find("wtorek") != mapaDni.end())
        cout << "wyraz 'wtorek' zostal znaleziony" << endl;
    if (mapaDni.find("czwartek") == mapaDni.end())
        cout << "wyraz 'czwartek' nie zostal znaleziono" << endl;
    return 0;
}
```

Zadanie 1 (25 pkt)

Problem do rozwiązania:

Trzeba napisać program, który umożliwi obsługę danych dotyczących studentów, które zostały scharakteryzowane następująco:

- Student opisany jest podstawowymi atrybutami takimi jak nr albumu, imię i nazwisko, aktualny semestr studiów. Studentów może być od 1 do n.
- Każdy ze studentów w danym momencie studiuje na określonym (1 z 7) semestrze, gromadzi również historię o semestrach na których studiował wcześniej.

- W każdym z semestrów student ma do zaliczenia szereg przedmiotów (od 1 do n).
- Każdy z przedmiotów może być oceniony, każda z ocen składa się ze stopnia (od 2 do 5), daty wpisu oraz imienia i nazwiska prowadzącego.

Przez obsługę danych należy rozumieć ich dodawanie, usuwanie, edycję, przeglądanie, wyszukiwanie itp. Konieczny jest oczywiście odpowiedni „program główny”, który pozwoli te operacje wykonywać

Proszę wykorzystać różne struktury danyc h z biblioteki STL aby zrealizować tę aplikację (jakie komu wygodnie).