C++ programavimo kalba

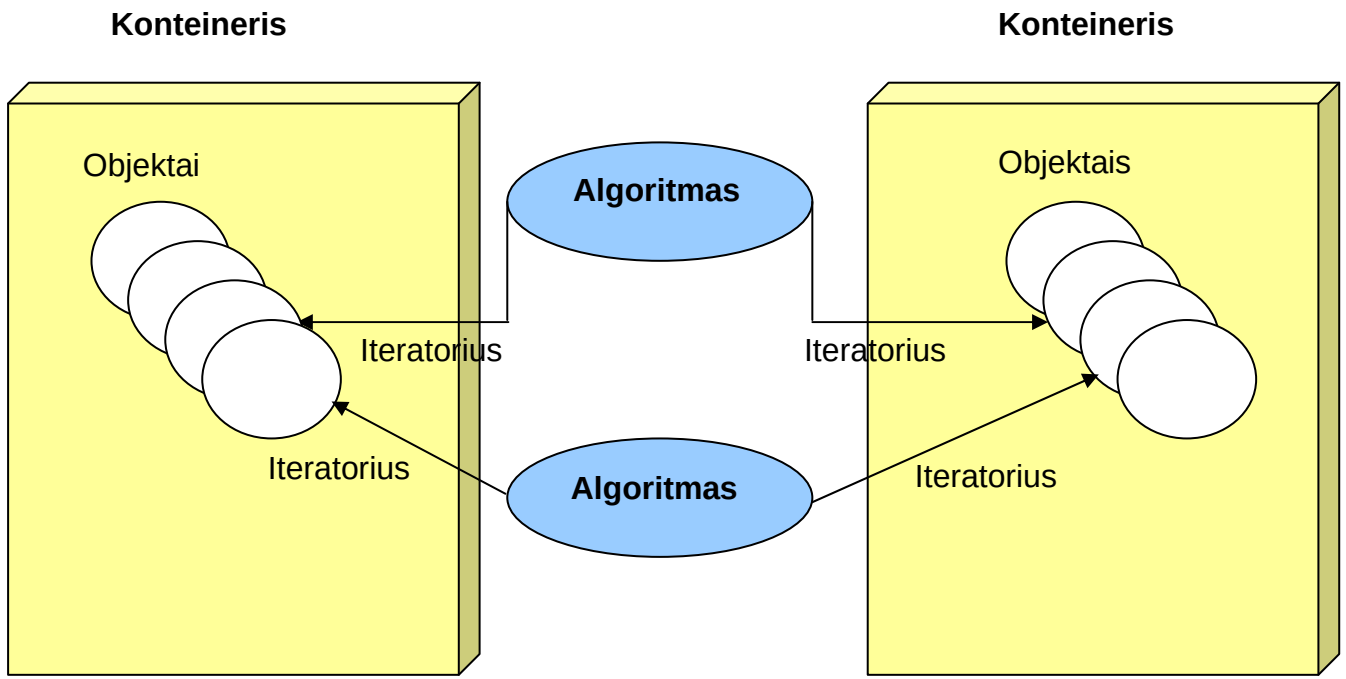
Standartinė šablonų biblioteka (STL) – tęsinys

(12 paskaita)

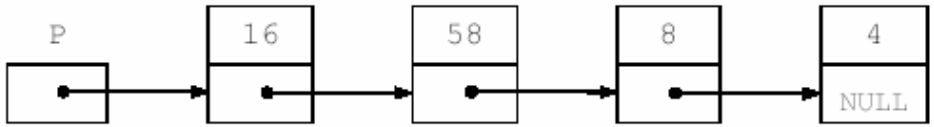
Standartinė šablonų biblioteka (STL)

- **STL** – tai konteinerinių (saugojimo) klasių (t.y. vektorių, sąrašų, eilių, stekų) ir bendro pobūdžio algoritmų (rūšiavimo, paieškos, kopijavimo) biblioteka.
- **STL** sudaro:
  - **Konteineriai** – tai būdas, kaip duomenys yra saugomi atmintyje. Duomenys gali būti tipiniai (*int, float...*) arba objektai. Konteineriai grupės;
    - Nuoseklūs konteineiriai
    - Asociatyviniai konteineriai
  - **Algoritmai** – tai procedūros (funkcijos), kurios taikomos konteinerio duomenų apdorojimui. Šios funkcijos nepriklauso klasėms;
  - **Iteratoriai** – tai apibendrintos rodyklės, jungiančios algoritmus ir konteinerių elementus. Didinant ar mažinant iteratoriaus reikšmę, pasiekiamas vis kitas konteinerio elementas.

Konteineriai, algoritmai, iteratoriai



Duomenų struktūra



Baziniai nuoseklūs konteineriai

	Charakteristikos	Privalumai ir trūkumai
C++ masyvas	Fiksuotas dydis	1. Greita paieška (pagal indekso numerį). 2. Sudėtingai ir lėtai įterpiamas arba ištrinamas vidurinis masyvo elementas. 3. Masyvo dydis negali kisti programos vykdymo metu.
Vektorius	Plečiamas masyvas	1. Greita paieška (pagal indekso numerį). 2. Sudėtingai ir lėtai įterpiamas arba ištrinamas vidurinis masyvo elementas. 3. Lengvai trinamas ar įterpiamas elementas paskutinis masyvo elementas.
Sąrašas	Dvikrypčiai sąrašai	1. Greitai įterpiamas ar naikinamas bet kuris elementas. 2. Greitai randamas pirmas ir paskutinis elementai 3. Lėta atsitiktinio elemento paieška.
Dekai	Panašus į vektorių tik galimas priėjimas (modifikavimas) iš bet kurio galo	1. Greita atsitiktinio elemento paieška. 2. Sudėtingai ir lėtai įterpiamas arba ištrinamas vidurinis masyvo elementas. 3. Greitai trinamas ar įterpiamas pirmas ir paskutinis elementai

Bendri visiems konteineriams metodai

Metodas	Aprašymas
size()	Grąžina konteineryje esančių elementų skaičių
empty()	Grąžina *true*, jei konteineris tuščias
begin()	Grąžina konteinerio pradžios iteratorių
end()	Grąžina konteinerio pabaigos iteratorių

Konteinerių modifikacijos (adapters)

Konteineris	Implemetacija	Charakteristika
Stack (stekai)	Padaroma iš vektorių, sąrašų ir dekų	Galima skaityti tik vėliausiai įrašytą elementą (struktūra LIFO – last in, first out). Įterpiamas ir šalinama elementas tik iš galo.
Queue (eilės)	Padaroma iš sąrašų ir dekų	Nauji elementai prijungiami sąrašo gale, o skaitymui yra prieinamas tik pirmasis elementas (struktūra FIFO – first in, first out);
Priority_stack (prioritetinės eilės)	Padaroma iš vektorių ir dekų	Įterpiamas elementas į bet kurį galą, o ištrinamas iš išrūšiuoto konteinerio iš kito galo.

Stekai

Stekai – viena iš populiariausių duomenų struktūrų. Duomenys organizuojami pagal principą LIFO , tai reiškia, kad steko elementus galima pridėti (*push*) ir pašalinti (*pop*) tik iš vieno galo. Šis galas vadinamas viršūne.

STL apibrėžta šabloninė klasė stack. Jos pajungimui reikalingas antraštės failas stack.h
Stekai nėra nepriklausoma konteinerinė klasė. Tai konteinerių adapteris, kuris modifikuoja bazinius konteinerius (pagal nutylėjimą dekus). Klasė stack tik leidžia konteineriams elgtis kaip stekui.

// sukuriamas stekas iš deko konteinerio
stack< deque <int> > aStack;
stack<int> stackOne;

// sukuriamas stekas iš sąrašo konteinerio
stack< Customer, list<Customer> > custom_stack

Eilės

Eilės – taip pat viena iš populiariausių duomenų struktūrų. Duomenys organizuojami pagal principą FIFO (first in, first out), tai reiškia, kad nauji elementai pridedami iš vieno galo (uodegos), o šalinami iš kito galo (pradžios). Kaip ir stekai, eilės – tai bazinių konteinerių modifikatoriai. Eilės bazinis konteineris pagal nulylėjimą – dekas.
STL apibrėžta šabloninė klasė queue. Jos pajungimui reikalingas antraštės failas queue.h

```
// sukuriama eilė iš deko konteinerio
queue<float> queueTwo;

// sukuriama eilė iš sąrašo konteinerio
queue<int, list<int> > queueOne;
queue<Customer, list<Customer> > queueThree;
```

Pavyzdys

```
#include <queue>
#include <iostream>

using namespace std;

int main ()
{queue<int> q;                          // sudaroma eilė
  q.push(1); q.push(2);                 // pildoma elementais
      cout<< q.front() << endl;
  q.pop();                              // šalinamas pirmas elementas
      cout<< q.front() << endl;
  q.pop();
return 0; }
```

Atsakymas:

1
2

Iteratoriai

Iteratoriai – tai rodyklių (pointer) vaidmenį atliekantys objektai, kurių pagalba galima pasiekti tam tikrą konteinerio elementą. Dažnai jie naudojami tam, kad nuosekliai (iteratyviai) prisirišti prie vis kito elemento, tai reiškia, kad iteratorių paskirtis surišti konteinerius ir algoritmus.
STL iteratoriams sukurti naudojama klasė iterator.

Naudojami trijų pagrindinių klasių iteratoriai:

- Judantys į priekį (forward)
- Judantys abiejomis kryptimis (bidirectional)
- Atsitiktiniai (random)

Specializuoti iteratoriai darbui su I/O:

- Įvedimo (rodo į įvedimo įrenginį t.y. *cin* arba *failas*)
- Išvedimo (rodo į išvedimo įrenginį t.y. *cout* arba *failas*)

Konteinerio elemento reikšmė gaunama naudojant “*” simbolį, pvz. *value =*iter*;

Iteratoriai

	Vector	List	Deque	Set	Multiset	Map	Multimap
Random	X		X				
Bidirectional	X	X	X	X	X	X	X
Forward	X	X	X	X	X	X	X
Input	X	X	X	X	X	X	X
Output	X	X	X	X	X	X	X

STL automatiškai parenka iteratorių konteineriui.

Iteratoriai

Algoritmai taip pat reikalauja tam tikro iteratoriaus tipo, todėl turi būti išlaikytas iteratoriaus suderinamumas iš vienos pusės su algoritmu iš kitos pusės su konteineriu (objektu).
Aukštesnio lygio iteratorius tinka algoritmui, reikalaujančiam žemesnio lygio iteratoriaus.

```
list <int> iList;
list <int>::iterator iter;
```

```
#include <iostream>
#include <list>
#include <algorithm>
void main()
{  int arr[ ] = {2, 3, 4, 5};
   list<int> theList;
      for (int k=0; k<4; k++)
           theList.push_back ( arr[k] );        // užpildomas sąrašas elementais
   list<int>::iterator iter;
       for (iter = theList.begin(); iter != theList.end(); iter++)
            cout<< *iter << ‘  ‘; }
```

Algoritmai

SLT saugoma apie 60 universalių algoritmų, kurie atlieka rutinines operacijas su duomenimis (find(), search(), count(), sort(), merge(), transform().
Jie apibrėžti antraštės faile <*algorithm*> ir naudoja standartinę vardų erdvę std.

Pavyzdys:

```
#include <iostream>
#include <algorithm>

using namespace std;
int arr[ ] = {11, 22, 33, 44, 55, 66,77 };

void main()
{   int *ptr;
    ptr = find( arr, arr+8; 33);              // rasti pirma 33
    cout<<“Pirmasis 33” << (ptr- arr);
}
```

Asociatyviniai konteineriai

Nuosekliųjų konteinerių trūkumas – lėta paieška, einant nuo vieno elemento prie kito.

Asociatyviniuose konteineiriuose elementai nėra rikiuojami eilėje, bet siejami su **raktais,** kurie atlieka indekso funkcijas. Raktais dažniausiai būna skaičiai arba eilutės. Pagrindinis asociatyvinių konteinerių privalumas – l.greita paieška

STL apibrėžia dvi pagrindines asociatyvinių konteinerių grupes:

- *set (aibė)*
- *map (žemėlapis)*

Aibėse saugomi unikalūs raktai, žemėlapiuose – raktų ir reikšmių poros.

Aibėse ir žemėlapiuose raktai unikalūs, tačiau multiaibėse bei multižemėlapiuose raktai gali kartotis.

Norint naudoti asociatyvinius konteinerius, reikia prisijungti antraštės failus map.h arba set.h

Aibės

Aibės dažnai naudojamos vartotojo sukurtų objektų saugojimui. Pvz. darbuotojų duomenų bazė. Tačiau aibės gali saugoti ir paprastesnius elementus – eilutes.

```
#include <iostream>
#include <set>
#include <string>
#include <algorithm>

void main()
{ string names [ ] = {“Jonas”, Petras”, “Ona”, “Antanas”};   // string objektų masyvas
   set<string, less<string> > nameSet(names, names+4);  // aibė į masyvą
   set<string, less<string> >::iterator iter;                    // iteratorius
   nameSet.insert(“Jurgis”);
   nameSet.insert(“Mikas”);
   nameSet.erase(“Mikas”);

 cout<<“Size”<<nameSet.size()<<endl;
                                                                  // less –rūšiavimo algoritmas
```

Aibės

```
 iter = nameSet.begin();
      while (iter != nameSet.end() )
          cout<<*iter++<<“\n”;

String searchName;
cout << Iveskite ieskoma varda: “;
cin >> searchName;

iter = nameSet.find(searchName);
if ( iter == nameSet.end() )
   cout << “ Vardas “<< searchName << “nerastas” << endl;
else
   cout << “Vardas “<<*iter << “rastas…”<<endl;
}
```

Šiame pavyzdyje vykdoma sukuriama aibė, kurią sudaro eilutės. Vykdoma elemento paieška, aibės papildymas ir elemento šalinimas.

Žemėlapiai (map)

Žemėlapiai – **tai poros, kurias sudaro *rakto objektas* ir *reikšmės objektas***.
Raktas – tai indeksas, pagal kurį randama reikšmė. Raktais dažniausiai būna skaičiai arba eilutės, bet tai gali būti ir objektai arba konteineriai.
Žemėlapiai kartais naudojami kaip asociatyviniai masyvai. Prijungimas <map>.

```
#include <iostream>
#include <map>
#include <string>

void main()
{ string name;  int pop;
  string states[ ] = {“Montana”, “Arizona”, “Nevada”, “Colorado”, “Florida”};
  int pops[ ] = {470, 280, 787, 985, 562};
  map <string, int, less<string> > mapStates;          //map
  map <string, int, less<string> > :: iterator iter;   //iterator


 // string – eilute(raktas), int - reikšmė,  less – išrūšiavimas mažėjimo tvarka
```

Žemėlapiai (map)

```
for  (int j = 0; j<6; j++)
{  name = states[ j ];
   pop = pops[ j ];
   mapstates [name] = pop;
}
  cout << “Įveskite valstiją: “;   cin >> name;
  cout << “Gyventojų skaičius “<<mapStates[name]<<endl;

  for (iter = mapStates.begin();  iter != mapStates.end();  iter++ )
      cout << (*iter).first <<‘  ‘ <<(*iter).second <<endl;

}


//  (*iter).first  išveda raktą, (*iter).second - reikšmę
```