C++ programavimo kalba

Standartinė šablonų biblioteka (STL)

Duomenų struktūros

(11paskaita)

Šablonai

Programuojant egzistuoja situacijos, kai reikia atlikti tuos pačius veiksmus su skirtingais duomenų tipais (pvz. modulio radimas, kėlimas laipsniu).
Tam patogu naudoti **šablonines** arba **parametrizuotas** funkcijas ir klases.

Funkijos šablono sintaksė

```
template <typename identifier> funkcijos_deklaravimas
template <class identifier> funkcijos_deklaravimas
```

Klasės šablono sintaksė

```
template <class Tipas> class klasės_pavadinimas
{  };
```

Standartinė šablonų biblioteka (STL)

C++ klasės sudaro puikią galimybę klasių pagalba kurti **duomenų struktūrų** bibliotekas, dėl to C++ standarte yra įtraukta konteinerinių klasių biblioteka. Siekiant platesnio tokių klasių pritaikomumo, klasės bibliotekoje apibrėžtos kaip šabloninės, todėl biblioteka buvo pavadinta **standartine šablonų biblioteka (Standard Template Library)**.

STL autoriai A.Stepanov ir M.Lee (Hewlett Packard).

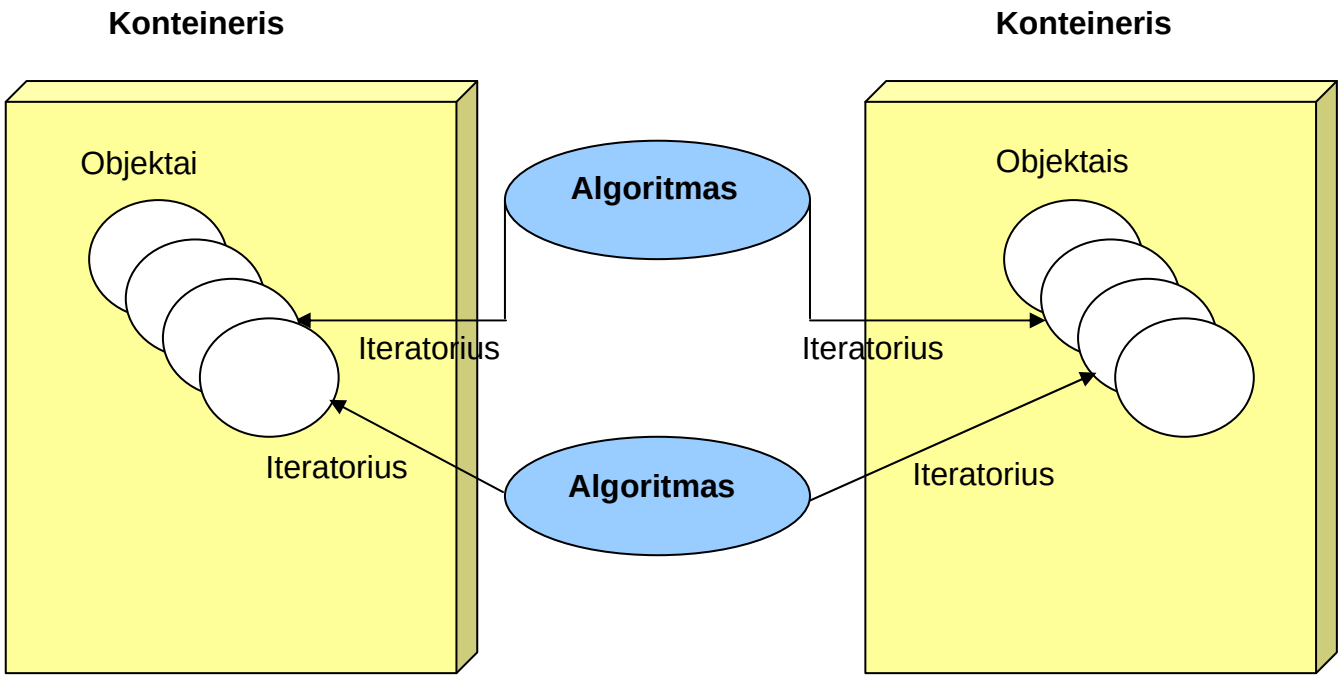
- STL – tai konteinerinių (saugojimo) šabloninių klasių (vektoriai, sąrašai, eilės, stekai, dekai) ir bendro pobūdžio šabloninių funkcijų-algoritmų (rūšiavimo, paieškos, kopijavimo) biblioteka.
- STL įtraukta į C++ standartą ir platinama nemokamai.
- STL tikslas – taupyti programuotojų darbą ir suteikti galimybę naudotis jau sukurtais ir patikimai veikiančiais klasių bei funkcijų šablonais.

Standartinė šablonų biblioteka (STL)

STL branduolį sudaro:

- **Konteineriai** – tai klasės, skirtos duomenims saugoti kompiuterio atmintyje. Duomenys gali būti tipiniai (int, float...) arba objektai. *Konteineriai – tai šabloninės klasės*, kurios gali saugoti be kokio tipo duomenis. Konteinerių tipai:
  - Nuoseklūs konteineiriai;
  - Asociatyviniai konteineriai.
- **Algoritmai** – tai procedūros (funkcijos), kurios taikomos konteinerio duomenų apdorojimui pvz. rūšiavimui, kopijavimui, paieškai. *Algoritmai – tai šabloninės funkcijos*. Šios funkcijos nepriklauso konteinerinėms klasėms;
- **Iteratoriai** – tai apibendrintos rodyklės, jungiančios algoritmus ir konteinerių elementus. Didinant ar mažinant iteratoriaus reikšmę, pasiekiamas vis kitas konteinerio elementas.

Konteineriai, algoritmai, iteratoriai

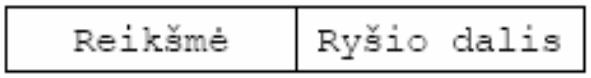


Duomenų sąrašai

Konteinerinės šabloninės klasės siejamos su duomenų sąrašais, todėl toliau bus pateikta bazinė informacija apie duomenų sąrašus.

Sąrašais vadinami tokie duomenų rinkiniai, kurių elementus apibūdina ne tik jų reikšmės, bet ir **tvarkymo taisyklės bei ryšiai tarp elementų**. Keičiant elementų tipus, ryšių tarp elementų būdus ir tvarkymo taisykles, galima sudaryti daugybę įvairių tipų sąrašų (eiles, stekus, dekus, medžius).

Paprasčiausi yra **statiniai sąrašai**, kurie gali būti realizuojami statiniuose masyvuose, papildant juos tvarkymo procedūromis ir užpildymo charakteristika. Tačiau universalesnė sąrašų sudarymo priemonė yra **dinaminės struktūros**, kurios sudarytos iš elementų su nuorodomis. Tokiuose elementuose saugomos ne tik jų reikšmės, bet ir ryšio su kitais elementais aprašymai. Taip charakterizuojami **dinaminiai sąrašai**.



Duomenų sąrašai

Iš elementų, kurių ryšio dalyje yra tik viena rodyklė, galima sudaryti tik **tiesinius sąrašus** dar vadinamus nuosekliai sujungtomis grandinėmis. Papildant tiesinius sąrašus tvarkymo procedūromis ir išorinio ryšio priemonėmis, galima sudaryti tokias tipines jų modifikacijas:

- **sąrašus**, kuriuose nėra apribojimų sąrašo elementų analizei, tvarkymui ir apdorojimui;
- **eiles**, kuriose nauji elementai prijungiami sąrašo gale, o skaitymui yra prieinamas tik pirmasis elementas (**FIFO – First In, First Out**);
- **stekus**, kuriuose galima skaityti tik vėliausiai įrašytą elementą (**LIFO – Last In, First Out**);
- **dekus**, kuriuose elementai gali būti rašomi ir skaitomi tiek sąrašo pradžioje, tiek gale (**DEQUE – Double-Ended QUEue**).

Tokios sąrašinės struktūros plačiai vartojamos ne tik taikomosiose, bet ir sisteminėse programose. Pavyzdžiui, stekuose yra saugomi pertraukiamų procesų parametrai, organizuojami lokalūs duomenys. Eilės yra populiari pagalbinių (buferinių) atminčių, kuriose kaupiami iš lėtų įrenginių gaunami arba į juos siunčiami duomenys, organizavimo priemonė.

Duomenų sąrašai

Kiekvieno sąrašo tipo realizavimui skirta dinaminė struktūra turi turėti jos elementus aprašančią struktūrą, rodyklę arba rodykles į išoriniams ryšiams skirtus elementus ir tokias tvarkymo operacijas:

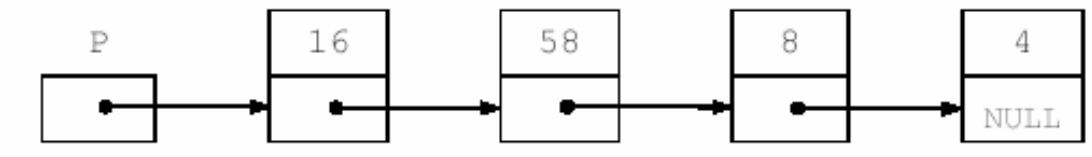
- naujo sąrašo sukūrimo;
- naujo elemento prijungimo;
- elemento pašalinimo;
- sąrašo skaitymo ir analizės;
- sąrašo tvarkymo.

Sudarant sąrašų tvarkymo procedūras, reikalingas papildomas susitarimas apie tai, kaip bus žymima sąrašo pabaiga ir kaip bus žymimas tuščias sąrašas. Yra priimta tiek sąrašo pabaigą, tiek tuščią sąrašą dinaminėse struktūrose žymėti nuline rodykle **NULL.**

Duomenų sąrašai gali būti realizuojami naudojant struktūras, tačiau C++ programavimo kalboje tam naudojamos klasės.

Stekas

Steko grafinis vaizdas parodytas stačiakampiais elementais, padalintais į tiek dalių, kiek yra laukų jo struktūroje: t.y. duomenų ir adreso. Duomenų lauke saugomos reikšmės. Rodyklės tipo lauke saugomas adresas (nuoroda) į kitą sąrašo elementą.
Nuoroda pavaizduota linija su rodykle gale. Tos linijos pradžia yra nuorodos lauko viduje. Rodyklė remiasi į elementą vaizduojantį stačiakampį bet kurioje vietoje. Jeigu nuoroda neegzistuoja, tuomet linija nėra brėžiama ir laukas lieka tuščias.



Steko programos pavyzdys

```
#include <iostream>
#include <fstream>
using namespace std;
            // Struktūros tipo apibrėžimas
  typedef struct list { int sk; struct list *next; } sar;

            // Klasės Dinsar apibrėžimas
class DinSar {
                    sar *P;                          // Sąrašo pradžia
          //struct list { int sk; struct list *next; } *P;     - galima ir taip

                public:
                    DinSar();                        // Konstruktorius
                    ~DinSar();                       // Destruktorius
                    void Formuoti(ifstream);         // Formuoja sąrašą
                    void Elementas(int);             // Prijungia elementą
                    void Spausdinti();               // Spausdina sąrašą
};
```

Steko programos pavyzdys

```
// Pagrindinė programa
void main() {
ifstream D;
DinSar A;                          // Klasės DinSar objektas
D.open("Duom.dat");
   if (D == NULL) {
       cout << "Failo 'Duom.dat' nepavyko atidaryti";
       exit (1);
    }
A.Formuoti(D);                     // Formuojamas sąrašas
D.close();
A.Spausdinti();                    // Spausdinamas sąrašas
A.~DinSar();
}
```

Steko programos pavyzdys

```
// Konstruktorius
DinSar::DinSar() { P = NULL; }

// Destruktorius
DinSar::~DinSar() {
sar *D = P;
while(P != NULL) {
D = P;     // Rodyklė į naikinamą elementą
P = P->next;   // Kito elemento adresas
delete( D ); }   // Elemento naikinimas
}

// Formuoja netiesioginį sąrašą
void DinSar::Formuoti(ifstream F) {
int k;
while(F.eof()) {
  F >> k;
Elementas(k); }
}
```

```
// Prijungia elementą
void DinSar::Elementas(int Sk) {
sar *R;
R = new sar;        // Naujo elemento sukūrimas
R->sk = Sk;         // Užpildymas duomenimis
R->next = P;        // Prijungimas prie sąrašo
P = R;              // Sąrašo pradžios pakeitimas
}

// Sąrašo spausdinimas
void DinSar::Spausdinti() {
sar *D = P;
while(D != NULL) {
cout << D->sk << " ";   // Reikšmės spausdinim.
D = D->next; }              // Kito elemento adresas
cout << "\n";
}
```

STL konteineriai (saugojimo klasės)

Konteineriai (saugojimo klasės) – tai šabloninės klasės, skirtos duomenų saugojimui, paieškai... Konteinerių privalumas – didesnis duomenų apdorojimo greitis. Masyvais C++ laikomas duomenų saugojimo būdas, kai jo elementų skaičius apibrėžiamas iš anksto. Pavyzdžiui: int A[10];

STL apibrėžia 7 konteinerių tipus, kurie skirstomi į dvi kategorijas: nuoseklūs ir asociatyviniai.

Nuoseklūs konteineriai (tiesiniai sąrašai)	Asociatyviniai konteineriai:
• vektoriai (vector) • sąrašai (list) • dekai (deque) • stekai (stack) • eilės (queue) • prioritetinės eilės (priority_queue)	• map • multimap • set • multiset

Nuoseklūs konteineriai (vektoriai)

Nuosekliuose konteineriuose saugomi duomenys gali būti įsivaizuojami kaip pvz. linija namų gatvėje. Visi elementai turi savo poziciją bei kaimynus.

Vektoriai – tai konteinerinė klasė, kurios elementai pasiekiami pagal indeksą. Vektoriai panašūs į vienmačius masyvus, tačiau jie yra dinaminiai bei užtikrina indeksų kontrolę.

Vektorių klasė apibrėžta antraštės faile **vector.h** ir naudoja std vardų erdvę.

```
#include <iostream>
#include <vector>
using namespace std;
void main()
 {  vector<int> v;                    // sudaromas nulinio ilgio vektorius
      for (int i = 0; i<5; i++)
           v.push_back(i);             // papildomas vektorius elementu, vektoriaus gale
   cout<<“Vektoriaus dydis”<<v.size() << endl;
      for (i=0; i<v.size(); i++)
          cout<< v[ i ]<<endl;
 }
```

Pagrindinės vektorių funkcijos

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

void main()
{ double arr[ ]= {1.1, 2.2, 3.3, 4.4 };
  vector<double> v1 (arr, arr+4) ;          // sudaromas vektorius
  while ( ! v1.empty () )
    { cout<< v1.back() <<endl;              // išvedamas paskutinis elementas
      v1.pop_back();  }                     // pašalinamas paskutinis elementas
  for (int i = 0; i<5; i++)
       v.push_back(i);                      // pridedamas elementas į galą
v1.insert (v1.begin()+2, 2.5)               // įterpiamas 2.5 po antrojo elemento
v1.erase (v1.begin()+2 )                    // ištrinamas trečias elementas

cout<<"Vektoriaus dydis"<<v.size() << endl;
   for (i=0; i<v1.size(); i++)              // pridedamas elementas į galą
       cout<< v1[i]<<endl;
}
```

Nuoseklūs konteineriai (sąrašai)

Sąrašai (lists) – tai nuoseklūs konteineriai, optimizuoti situacijoms, kai reikia įterpti arba pašalinti objektą. Pvz. turime darbuotojų sąrašą, saugomą abėcėlės tvarka.

Sujungti sąrašai (Linked list) – tai lanksti duomenų saugojimo būdas, nenaudojant masyvų. Čia naudojami struktūra kurioje saugomi duomenys ir rodyklės. Rodyklė saugo adresą sekančio sąrašo elemento.

Dvigubai sujungti sąrašai (Double linked list) – tai sąrašas, kuriame saugomos dvi rodyklės t.y. abiejų kaimyninių elementų adresai.

Sąrašų privalumas – neribojamas sąrašo dydis, greita paieška.

Sarašo klasė apibrėžta antraštės faile **list.h** ir naudoja std vardų erdvę.

Nuoseklūs konteineriai (sąrašai)

```
#include <iostream>
#include <list>
using namespace std;
void main()
{   list<int> ilist;
    ilist.push_back(30);                    // elementai talpinami sąrašo gale
    ilist.push_back(40);
    ilist.push_front(20);                   // elementai talpinami sąrašo priekyje
    ilist.push_front(10);
    ilist.push_front (10);
    ilist.unique()                           // pašalina dublikuotus elementus
    ilist.reverse()                         // sukeičia eiliškumą t.y. 40, 30, 20, 10
    cout<<“Saraso dydis”<<ilist.size() << endl;
    for (int i=0; i < ilist.size(); i++)
    {  cout<< ilist.front() <<endl;          // skaitomi elementai iš priekio
       ilist.pop_front();                    // trinami elementai iš priekio
     }
}
```

Dekai

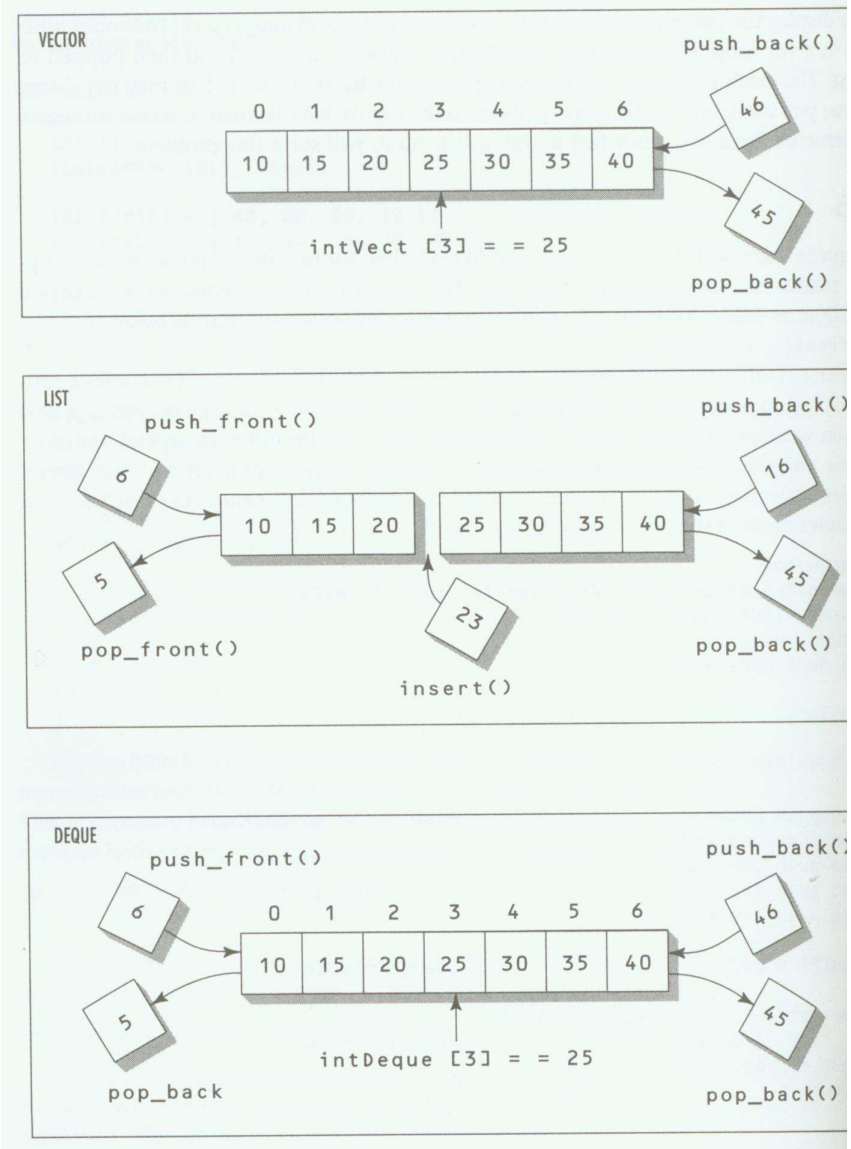
Dekai – tai vektorių ir sąrašų kombinacija. Į dekus, kaip ir į vektorius galima kreiptis per indeksą [ ] , tačiau dekų elementai kaip ir sąrašų gali būti pasiekiami tiek ir pradžios tiek ir iš galo. Dekas – tai double ended vector (Double Ended QUEue), palaikantys funkcijas *push_front(), pop_front(), front().*

Atminties rezervavimas skirtingas nei vektorių ar sąrašų. Vektoriai visada rezervuoja nuoseklų atminties segmentą. Dekai gali saugoti duomenis atskiruose segmentuose.

Dekų klasė apibrėžta antraštės faile **deque.h** ir naudoja std vardų erdvę.

```
void main()
{   deque<int> deq;
      deq.push_back(30);          // elementai talpinami deko gale
      deq.push_back(40);
      deq.push_front(20);
      deq.push_front(10);
     deq[2] = 33;                 // trečias deko elementas 33
for (int i=0; i < deq.size(); i++)
   {  cout<< deq[ i] <<endl; }     // išvedami elementai
```

Trumpai


VECTOR
push_back()
0 1 2 3 4 5 6
10 15 20 25 30 35 40
46
45
intVect [3] = = 25
pop_back()
LIST
push_front()
push_back()
6
5
10 15 20
25 30 35 40
16
45
23
pop_front()
pop_back()
insert()
DEQUE
push_front()
push_back()
6
0 1 2 3 4 5 6
10 15 20 25 30 35 40
46
5
45
intDeque [3] = = 25
pop_back
pop_back()