Standart Şablon Kütüphanesi (Standard Template Library-STL)

Dr. Metin Özkan

Giriş

- Birçok veri yapıları ve algoritmalar, genellikle kullanılmaktadır. C++ standart komitesi, bunları kapsayan standart şablon kütüphanesini (Standart Template Library, STL), C++ standart kütüphanesine ekledi.

- STL, genellikle kullanılan veri yapıları ve bu verileri işlemek için kullanılan algoritmaları gerçekleyen güçlü, şablon tabanlı, yeniden kullanılabilir bileşenleri tanımlamaktadır.

Giriş

- STL, üç ana bileşenden oluşur:
  - ***Container**: Veri gruplarını depolamaya yarayan şablon sınıflar*
    - *Her STL container, bir sınıf olduğu için üye fonksiyonları vardır.*
    - *Örnek container olarak, `vector` (dinamik olarak boyutu değişen dizi, dynamically resizable array), list (çifte bağlı liste, doubly linked list) verilebilir.*
  - ***Iterator**: Container içinde saklanan verilere erişmek için tanımlanmış genel gösterge (pointer)*
    - *Göstergelerinkine benzer özellikleri vardır. Algoritmalar, işlem yürütmek için iterator kullanmaktadır.*
  - ***Algorithm**: Container içindeki veriler üzerinde işlem yapabilen genel fonksiyon şablonlar*
    - *Arama, sıralama, elamanları karşılaştırma gibi veri işlemleri yürüten fonksiyonlardır.*
    - *Bu fonksiyonlar, container içindeki verilere erişimde iterator kullanmaktadır.*

Containers

- Konteynerler (Containers), dört başlıca kategoride toplanabilir.
  - *Ardışık konteynerler (Sequence containers)*
    - *array: Sabit boyutlu, elemanlara doğrudan erişim*
    - *deque: baştan ve sondan hızlı ekleme ve çıkarma, elemanlara doğrudan erişim*
    - *list: çifte bağlı liste (doubly linked list), herhangi bir yere hızlı ekleme ve çıkarma*
    - *vector: Sondan hızlı ekleme ve çıkarma, elemanlara doğrudan erişim*
  - *Düzenli çağrışımlı konteyner (ordered associative container): Anahtarlar sıralı düzende tutulur.*
    - *set: Hızlı bakma, çok kopyaya müsaade yok*
    - *multiset: Hızlı bakma, çok kopyaya müsaade var*
    - *map: bire bir eşleme, çok kopyaya müsaade yok, hızlı anahtar tabanlı bakma*
    - *multimap: bire bir eşleme, çok kopyaya müsaade var, hızlı anahtar tabanlı bakma*

Containers

- Konteynerler (Containers), dört başlıca kategoride toplanabilir.
  - *Düzensiz çağrışımlı konteyner (unordered associative container):*
    - *unordered_set: Hızlı bakma, çok kopyaya müsaade yok*
    - *unordered_multiset: Hızlı bakma, çok kopyaya müsaade var*
    - *unordered_map: bire bir eşleme, çok kopyaya müsaade yok, hızlı anahtar tabanlı bakma*
    - *unordered_multimap: bire bir eşleme, çok kopyaya müsaade var, hızlı anahtar tabanlı bakma*
  - *Konteyner adaptörler (Container adapters)*
    - *stack: son giren ilk çıkar (last-in, first out, LIFO)*
    - *queue: ilk giren ilk çıkar (first-in, first out, FIFO)*
    - *priority_queue: En öncelikli eleman ilk çıkar*

Containers

- Konteyner ortak üye fonksiyonları
  - *default constructor: Boş bir konteyner başlatır.*
  - *copy constructor: Bir konteyneri bir başka konteynerin kopyası olarak yaratır.*
  - *destructor: konteyneri yok ederken, temizlenmesini sağlar.*
  - *empty: Konteynerde hiç eleman yoksa true, varsa false döndürür.*
  - *insert: Konternere, bir veri ekler.*
  - *size: Konternerdeki eleman sayısını döndürür.*
  - *operator=: Bir kontayneri bir başkasına kopyalar.*
  - *operator<, operator<=, operator>, operator>=, operator==, operator!= : İki konteyneri karşılaştırır, true ya da false döndürür.*
  - *swap : iki konteynerin elemanlarını yer değiştirir.*
  - *max_size: Bir konteyner için mümkün olan en fazla eleman sayısı döner.*

Containers

- Konteyner ortak üye fonksiyonları
  - *begin: Konteynerin ilk elemanı için iterator döndürür.*
  - *end: Konteynerin son elemanı için iterator döndürür.*
  - *rbegin: Konteynerin son elemanı için reverse_iterator döndürür.*
  - *rend: Konteynerin ilk elemanı için reverse_iterator döndürür.*
  - *erase: Bir veya daha çok elemanı konteynerden atar.*
  - *clear: Tüm elemanları, konteynerden atar.*

Iterators

- Farklı tiplerde iteratörler vardır:
  - *iterator: Konteyner elemanları gösteren bir iteratördür.*
  - *const_iterator: Konteyner elemanları gösteren bir iteratördür. Elemanları sadece okumak ve const işlemleri yürütmek için kullanılır.*
  - *reverse_iterator: Konteyner elemanları gösteren bir ters iteratördür. Konteynerde, tersten iterasyon için kullanılır.*
  - *const_reverse_iterator: Konteyner elemanları gösteren bir iteratördür. Elemanları sadece okumak ve const işlemleri yürütmek için kullanılır. Konteynerde, tersten iterasyon için kullanılır.*

Vector

- Vector, verileri ardışık bellek lokasyonlarında tutar.
  - *Elemanlara konum indeksi ile erişim vardır (sabit zamanda)*
  - *İterasyon her iki yönde yapılabilir (lineer zamanda)*
  - *Sondan eklemek ve çıkarmak hızlıdır (sabit bir zamanda)*

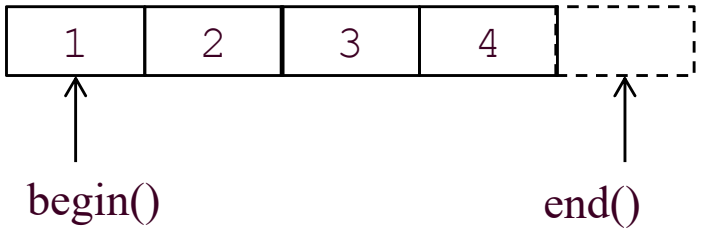
```
#include <iostream>
#include <vector>
using namespace std;

int main() {
          vector<int> v1;        //boş
          vector<double> v2(10);//10 elemanlı
          vector<int> v3(10,4); //10 elemanlı
                                 //değerleri 4 olan
          vector<int> v4(v3); //v3'ün kopyası olan

          //v1'nin sonuna 0'dan 4'e kadar değer ekle.
          for(int i=0;i<5;i++)
                    v1.push_back(i);
          //v1'nin değerlerini ekrana yazdır
          for(int i=0;i<5;i++)
                    cout<<v1[i]<<" ";
          //v2'ye 0'dan 9'a kadar değer yaz.
          for(int i=0;i<10;i++)
                    v2[i]=i;
          //v2'nin sondan 5 elemanını yaz, sonra sil.
          for(int i=0;i<5;i++){
                    cout<<v2.back();
                    v2.pop_back();
          }
          //v2'nin boyutunu ekrana yazdır.
          cout<<"Size of v2 is"<<v2.size();
          return 0;
}
```

List

- List, verileri bağlı liste (linked list) olarak, farklı bellek lokasyonlarına dağılabilir.
  - *Herhangi bir yere etkin ekleme ve çıkarma yapılabilir (sabit zamanda)*
  - *Elemanlar blok olarak, aynı ya da farklı konteynerlerde taşınabilir (sabit zamanda)*
  - *İterasyon her iki yönde yapılabilir (lineer zamanda)*




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

int main() {
        list<int> v1;          //boş
        int arr[]={1,2,3,4};
        list<int> v2(arr,arr+4);//arr' daki
                //değerler ile 4 elemanlı
        list<int>::iterator iter;

        //v1'nin sonuna ve başına 0'dan 4'e kadar
        //değer ekle.
        for(int i=0;i<5;i++){
                v1.push_back(i);
                v1.push_front(i);
        }
        //v1'nin değerlerini ekrana yazdır
        for(iter=v1.begin();iter!=v1.end();iter++)
                cout<<(*iter)<<" ";

        //v2'nin baştan 5 elemanını yaz, sonra sil.
        for(int i=0;i<5;i++){
                cout<<v2.front();
                v2.pop_front();
        }
        return 0;
}
```

List

```
#include <iostream>
#include <list>
#include <vector>
#include <iterator>
using namespace std;
int main() {
          list<int> mylist;
          list<int>::iterator iter;

          // set some initial values:
          for (int i=1; i<=5; ++i)
                                mylist.push_back(i);  // 1 2 3 4 5

          iter = mylist.begin();
          ++ iter;    // iter points now to number 2   ^

          mylist.insert (iter,10);                  // 1 10 2 3 4 5

          // "iter" still points to number 2                ^
          mylist.insert (iter,2,20);                // 1 10 20 20 2 3 4 5

          --iter;   // iter points now to the second 20      ^

          vector<int> myvector (2,30);
          mylist.insert (iter,myvector.begin(),myvector.end());
                                                    // 1 10 20 30 30 20 2 3 4 5
                                                    //                ^
          cout << "mylist contains:";
          for (iter=mylist.begin(); iter!=mylist.end(); ++iter)
                    cout << ' ' << *iter;
          cout << '\n';
          return 0;
}
```

Iterators (Reverse)

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

int main(){
            int num[]={1,2,3,4};
            string str[]={"ali", "veli"};

            vector <int> vec(num, num+4);    //initialize
            list <string> li(str, str+2);    //initialize

            vector <int>::iterator vecIter;
            list <string>::reverse_iterator liIter; //reverse iterator

            liIter=li.rbegin();              //iterate backwards
            while(liIter!=li.rend())         //through list
                        cout<<*liIter++<<' ';
            return 0;
}
```

Algorithms

- Şablon (template) fonksiyonlardır.
- Herhangi bir sınıfın üyesi değiller.
- Diziler (arrays), ile de kullanılabilir.
- Örnek olarak ele alınacak algoritma fonksiyonu,
  - `iterator find(iterator first, iterator last, Type const &value)`

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

int main(){
            int num[]={1,2,3,4};

            list <int> li(num, num+4);        //initialize

            list <int>::iterator liIter;      //forward iterator

            liIter=find(li.begin(), li.end(), 3); //look for 3
            if(liIter!=li.end())
                        cout<<"Found\n";
            else
                        cout<<"not found\n";
            return 0;
}
```

Bir Örnek - Çok işlev (Polymorphism) ile vector kullanımı

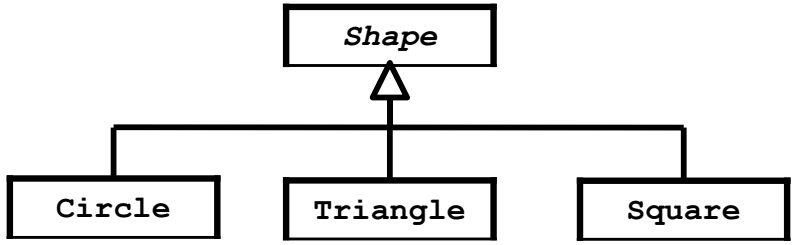
```
#include <vector>
#include <iostream>
using namespace std;

class Shape {
public:
          virtual void draw() = 0;
          virtual ~Shape() {};
};

class Circle : public Shape {
public:
          void draw() { cout << "Circle::draw\n"; }
          ~Circle() { cout << "~Circle\n"; }
};

class Triangle : public Shape {
public:
          void draw() { cout << "Triangle::draw\n"; }
          ~Triangle() { cout << "~Triangle\n"; }
};

class Square : public Shape {
public:
          void draw() { cout << "Square::draw\n"; }
          ~Square() { cout << "~Square\n"; }
};
```



Bir Örnek - Çok işlev (Polymorphism) ile vector kullanımı

```
#include <vector>
#include <iostream>
using namespace std;
typedef vector<Shape*> Container;
typedef vector<Shape*>::iterator Iter;
int main() {
            Container shapes, shapes1;
            shapes.push_back(new Circle);
            shapes.push_back(new Square);
            shapes.push_back(new Triangle);
            shapes1.push_back(new Circle);
            cout<<"Shape:\n";
            for(Iter i = shapes.begin();i != shapes.end(); i++)
                        (*i)->draw();
            cout<<"Shape1:\n";
            for(Iter i = shapes1.begin();i != shapes1.end(); i++)
                        (*i)->draw();
            shapes.swap(shapes1);
            cout<<"After the algorithm swap:\n"; cout<<"Shape:\n";
            for(Iter i = shapes.begin();i != shapes.end(); i++)
                        (*i)->draw();
            cout<<"Shape1:\n";
            for(Iter i = shapes1.begin();i != shapes1.end(); i++)
                        (*i)->draw();
            // ... Sometime later:
            cout<<"Free for Shape:\n";
            for(Iter j = shapes.begin();j != shapes.end(); j++)
                        delete *j;
            cout<<"Free for Shape1:\n";
            for(Iter j = shapes1.begin();j != shapes1.end(); j++)
                        delete *j;
}
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.