Standard Template Library dan Casting

IF2281- Pemrograman Berorientasi Objek

Dibuat Oleh Hananto

Standard Template Library (STL)

- STL adalah library standard sebagai bagian dari C++ berisi container class, iterator, algoritma yang bersifat generik.
- Diimplementasikan oleh Alexander Stepanov dan Meng Lee dari Hewlett Packard.
- Terdiri dari 5 jenis komponen:
  - Penampung(container), mengelola sekumpulan lokasi memori
  - Iterator: menyediakan sarana untuk iterasi terhadap suatu penampung
  - Objek fungsi: membungkus sebuah fungsi dalam objek yang digunakan komponen lain
  - Algoritma: mendefinisikan prosedur komputasi
  - Adaptor: mengadaptasikan komponen agar menyediakan interface berbeda

Penampung(Container)

- Penampung dalam STL dibagi menjadi
  - Sequence container: menyimpan kumpulan objek dalam susunan liner.contoh: Vector,Deque,List
  - Assosiative container: menyimpan kumpulan objek dengan fasilitas pengambilan objek(retrieval) berdasarkan nilai kunci. Contoh: Set, MultiSet,Map,MultiMap

Vector

- Vector adalah sequence dengan karakteristik:
  - Number of elements is dynamically
  - Support random access elements
  - Constant time insertion and removal last element
  - Linear time insertion and removal first or middle element
- Contoh:

```
Vector<int> V;
V.insert(v.begin(),3);
```

Deque

- Deque adalah sequence mirip dengan vector dengan karakteristik:
  - Number of elements is dynamically
  - Support random access elements
  - Constant time insertion and removal last element
  - Linear time insertion and removal middle element
  - Constant time insertion and removal first element
- Contoh:

```
deque<int> Q;
Q.push_back(3);
Q.push_front(1);
Q.insert(Q.begin()+1,2);
```

List

- List adalah adalah double linked list dengan karakteristik:
  - Support both forward and backward traversal
  - Constant time insertion and removal of first, middle and last element.
- Contoh:

```
List<int> L;
L.push_back(0);
L.push_front(1);
L.insert(++L.begin(),2);
```

Set dan MultiSet

- Set adalah assosiative sequence yang menyimpan objek berdasarkan kunci yang kita definisikan dari data yang disimpan.
- MultiSet identik dengan Set tetapi element dalam multiSet memungkinkan tidak unik

Map dan MultiMap

- Map adalah sequence yang menyimpan objek dengan kunci pengaksesan yang kita definisikan
- MultiMap identik dengan Map tetapi dalam MultiMap memungkinkan data yang disimpan tidak unik.

Iterator

- Iterator merupakan pointer C++ yang generik sehingga programmer dapat memanipulasi penampung dengan seragam.
- Dibagi menjadi:
  - Forward Iterator
  - Bidirectional Iterator
  - Random Access Iterator
  - Input dan Output Iterator

Objek Fungsi

- Objek yang mendefinisikan fungsi operator
- Dibagi menjadi:
  - Fungsi aritmatika: `minus, plus, times, divides, modulus, negate`
  - Fungsi perbandingan: `equal_to, not_equal_to, greater, less, greater_equal`
  - Fungsi logika: `logical_and,logical_or, logical_not`

Algoritma

- Implementasi algoritma yang melakukan manipulasi terhadap data container secara generik
- Contoh: `for_each(), find(), find_if(), count(),transform(),` …

Adaptor

- Spesialisasi dari container yang telah didefinisikan dengan perilaku yang spesifik.
- Contoh:
  - Stack: spesialiasi vector yang hanya memungkinan melakukan push dan pop objek
  - Queue: spesialisasi queue dimana insert hanya boleh diakhir dan remove hanya boleh diawal.

Casting(1)

- Konversi suatu type kedalam type lain
- Terdiri dari
  - `reinterpret_cast`: casting terhadap setiap type pointer, memungkinkan casting suatu pointer ke suatu integer dan sebaliknya
  - `static_cast`: casting terhadap suatu class ke kelas turunan atau sebaliknya
  - `dynamic_cast`: casting untuk polymorphic class
  - `const_cast`: casting terhadap suatu deklarasi objek statik

Casting(2)

- Reinterpret_cast

```
class A {};
class B {};
A * a = new A;
B * b = reinterpret_cast<B*>(a);
```

- Static_cast

```
class Base {};
class Derived: public Base {};
Base * a = new Base;
Derived * b = static_cast<Derived*>(a);
```

Casting(3)

- Dynamic_cast(1)

```
class Base { virtual dummy(){}; };
class Derived : public Base { };
Base* b1 = new Derived;
Base* b2 = new Base;
Derived* d1 = dynamic_cast<Derived*>(b1); // succeeds
Derived* d2 = dynamic_cast<Derived*>(b2); // fails:
  returns NULL
```

- Dynamic_cast(2)

```
class Base { virtual dummy(){}; };
class Derived : public Base { };
Base* b1 = new Derived;
Base* b2 = new Base;
Derived d1 = dynamic_cast<Derived&*>(b1);   // succeeds
Derived d2 = dynamic_cast<Derived&*>(b2); // fails:
// exception thrown
```

Casting(4)

- Const_cast

```
class C { . . .};
const C * a = new C;
C * b = const_cast<C*> (a);
```