Lecture 4 : Standard Template Library의 활용

현대 SW의 특징은 그 크기와 복잡도가 이전 Software와는 비교할 수 없을 정도로 커진 것이다. 이렇게 큰 SW는 이전과 같이 개인적인 또는 몇 사람의 작업으로 완성될 수는 없다. SW 개발에서 가장 중요한 덕목은 안정적인 구조를 가지는 것이다. 즉 오류(error)가 없어야 하며 만일 그런 오류가 발생한다고 하더라고 쉽게 대응할 수 있도록 SW가 구성되어 있어야 한다.

현대 공산품의 특징은 그 공산품을 구성하고 있는 부품까지를 최종 생산자가 만들지 않는다고 하는 것이다. 예를 들어 아이폰을 만들 때 패널과 메모리, 포장 용기(case) 등등은 다른 업체의 제품을 사들이어 조립하는 식으로 진행된다. 이렇게 생산을 가장 기본적인 단위가 아니라 이미 구성되어 활용되고 있는 중간 반제품 단계로부터 시작하는 것의 첫 번째 이유는 안정성 확보에 있다. 예를 들어 보자. 만일 거주할 집(house)을 하나 짓는다고 할 때 그 벽돌 하나하나까지 직접 만들어 사용하는 곳은 어디에도 없다. 대부분 이미 그 안정성이 확보된 벽돌을 사들이어 이를 쌓아서 집을 만드는 것이다. 이렇게 만들면 안정적인 벽을 만드는 것도 가능하고 또한 그 개발 기간도 단축할 수 있다. SW도 다르지 않다.

이미 그 안정성이 충분히 검증된 기본 함수가 있다고 한다면 개발을 source code level이 아니라 이 함수 단위부터 시작하면 더 빠르게 더 안정적인 SW를 개발하는 데 큰 도움을 받을 수 있다. 이런 취지로 시작된 일의 한 가지가 바로 Standard Template Library(STL)이다. 실제 대부분의 C++기만 개발 현장에서는 STL을 사용하며 대부분의 C++ 컴파일러를 설치하면 기본적으로 STL이 설치되므로 이는 C++ 기본 환경으로 이미 정착되었다. 현대 SW에서 C++로 하는 모든 작업은 결국 STL을 얼마나 잘 활용하는가의 문제로 귀납된다.

4.1 프로그래밍 패러다임 변화

- 특별한 소수의 사람만이 가능한 프로그래밍 (어셈블리 프로그래밍)
- High Level 언어의 등장 (Fortran)
- 사용자 편의 환경 (GUI)
- 공동작업을 위한 프로그래밍 환경 (소프트웨어 공학의 등장)
- 사용자 편의성이 더 강조된 언어
- 다양한 자료구조의 제공
  - STL, BOOST, LEDA
  - Python Package
- 다양한 Component Ware
- 확률 프로그래밍
- 병렬 프로그래밍
- 인공지능 기능 Platform
  - Tensorflow, Pytorch

4.2 STL (Standard Template Library)의 개발 과정

- C와 C++의 탄생 이유
- UNIX와 C언어
- C++를 위한 표준 라이브러리 (ANSI/ISO C++ 위원회의 승인통과)의 일부분이지만
- 따로 탐구되고 있는 독립적인 component라고 볼 수 있다.
- 좋은 사이트는 많지만 http://www.cplusplus.com/reference/ 를 추천함.

Q) STL과 같은 Component ware를 안 쓰고 끝까지 버티면 무슨 일이 벌어질까?

■ **우물론: 우물을 팔 때**

한 곳만을 골라 "깊이" 팔 것인가, 아니면 여러 군데를 조금씩 팔 것인가?

4.3 STL에 숨어있는 기본적인 철학

a. generic programming을 가능하게 한다.

sorting(숫자) = { sorting(정수), sorting(실수), sorting(string) }

예) 먹는다 = { 마신다, 삼킨다, 씹어 넘긴다.}

b. 효율성의 저하가 없는 데이터 추상화

c. Von Neumann 모델의 차용

d. Value Semantics 차용

STL을 사용하기 위해서 반드시 사전에 결정해야 할 사항

(a) 어떤 작업을 할 것인지

(b) 어느 정도의 시간을 사용할 수 있는지

(c) 프로그래밍의 어려움이 중요한지, 시스템 성능이 중요한지

4.4 STL의 기본적인 Architecture[1)]

- 컨테이너 (container) - 뭔가를 담는 통
- 알고리즘 (algorithm) - 담긴 것에 어떤 특정한 작업을 하는 함수
- 반복자 (iterator) - 담긴 자료를 어떤 순서로 훑어 내려가도록 해주는 일종의 특정 자료구조용 "pointer"
- 함수 객체 (function object) - 어떤 일을 하라고 지시할 수 있는 단위
- 할당기 (allocator) - 새로운 자료 공간을 만들어 사용하고 버리는 역할

Container

- 객체를 저장하는 자료구조. 클래스 템플릿으로 구현되어있다.
- sequence container, associative container로 나뉩니다.
  > Sequence Container = { array (C++ 11), vector, list, deque }

1) STL을 사용할 경우에는 C++ compiler의 version을 잘 확인해야 한다. 현재는 C++ 2020까지 나와 있으면 다음 version은 2023으로 예정되어 있다. https://en.cppreference.com/w/cpp/language/history

> Associative Container = { set, multiset, map, multimap }

Container class templates

Sequence containers:

array :	Array class (class template)
vector :	Vector (class template)
deque :	Double ended queue (class template)
forward_list :	Forward list (class template)
list :	List (class template)

Container adaptors:

stack :	LIFO stack (class template)
queue :	FIFO queue (class template)
priority_queue :	Priority queue (class template)

Associative containers:

set :	Set (class template)
multiset :	Multiple-key set (class template)
map :	Map (class template)
multimap :	Multiple-key map (class template)

Unordered associative containers:

unordered_set :	Unordered Set (class template)
unordered_multiset :	Unordered Multiset (class template)
unordered_map :	Unordered Map (class template)
unordered_multimap :	Unordered Multimap (class template)

Iterator

- 포인터와 비슷한 개념으로 컨테이너의 원소를 가리키고, 가리키는 원소에 접근하여 다음 원소를 가리키는 기능을 수행한다. Traversal(순회)이 가능하게 한다.

Algorithm

- 정렬, 삭제, 검색, 연산을 실행하는 일반화(generic)된 함수 템플릿.

Function Object

- 함수처럼 동작하는 객체로 operator() 연산자를 오버로딩 한 객체를 의미한다.

Container Adaptor

- 객체의 인터페이스를 수정하여 새로운 인터페이스를 제공하는 요소로 변경.

> Container Adaptor = { stack, queue, priority_queue }

어떤 경우에는 이것은 일반적인 container로 분류하기도 한다.

Allocator

- 컨테이너의 메모리 할당을 담당하는 클래스 객체로.
  모든 컨테이너는 각 객체별 전용 할당기를 가지고 있다.

4.5 Template와 Generic programming(일반형 프로그래밍)

```
int      i , j ;
float    f1, f2 ;
i  =  i +  j  ;
f1 = f1 + f2 ;
```

예) 어떤 데이터가 들어온다. 이것은 정수, 실수, char[10]일수도 있다. 이것을 정수 10자리, 실수 10.5 자리, %10c로 출력하는 프로그램을 작성하시오.

예) 어떤 데이터는 실수, 또는 같은 크기의 배열, 복소수 **a + bi** 가 될 수 있다. 이들을 더해서 그 결과 값을 return해주는 함수를 작성하시오.

4.5 Class Template와 function Template

```
template <typename T1, typename T2>
class pair {
  public :
       T1 first  ;
       T2 second ;
       pair(T1 x, T2 y) :
          first(x), second(y) { }
} ;

pair<int,char> mypir(13,'t') ;
pair<bool,                    double>
yourpair(true,0.1);
```

```
int u=3, v = 4 ;
double d = 0.132 , e = 3294.34 ;

template <typename T>
T getmax(T x, T y) {
  if ( x < y ) return y ;
  else return x ;
} // end of template function
getmax()

cout << getmax(u,v) << endl ;
cout << getmax(d,e) << endl ;
```

4.6 만일 복소수나 다른 자료구조에 대하여 비교(<)를 수행하고자 한다면 이를 따로 template 에 정의하여야 한다.

```
bool operator<(const point<int, int>& x , const point<int, int>& y ) {
int a , b ;
       a = x.first^2 + x.second^2 ;
       b = y.first^2 + y.second^2 ;
       return a < b ;
} // end of operator
```

4.7 배열자료 구조에서 지원하는 연산

일반 프로그래밍 언어	STL
생성 in O(1) Store in O(1) Retrieve in O(1)	여러분은 STL 컨테이너 중에서 어떤 자료구조가 어떤 연산을 지원하는지를 잘 숙지하고 있어야 한다.

4.8 STL vector를 이용한 프로그래밍의 예

```
template<typename C> void dump(const char *desc, C c) {
   cout.width(12);cout << left << desc << "==> ";
   copy(c.begin(),c.end(),ostream_iterator<typename \
   C::value_type>(cout," "));     cout << endl;
} // END OF DUMP FUNCTION( )


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

void main() {
 int ar[]={1,2,3,4,5,6,7,8,9};
    vector<string> v1;          dump("v1",v1);
    vector<double> v2(10);      dump("v2",v2);
    vector<int>    v3(10,7);    dump("v3",v3);
    vector<int>    v4(v3);      dump("v4",v4);
    vector<int>    v5(&ar[2],&ar[5]); dump("v5",v5);

} // END OF MAIN( )
```

```
#include <bits/stdc++.h>
#define allout(MSG,X) \
        cout<<"\n"<<MSG<<"\n";for(auto w:X)cout<<w<<", "

using namespace std;

int main () {

    int myarray[100] ; // C-style array
    vector<int> myv { 11, 22, 55, 30, 76, -25 } ;

    myv.push_back(-999);
    for(auto s : myv) cout << s << ",  " ;
    myv.pop_back();
    myv.pop_back();
    myv.pop_back();

    for(auto s : myv) cout << s << ",  " ;

  return 0;
} // end of main( )
```

결과 출력물

```
v1          ==>
v2          ==> 0 0 0 0 0 0 0 0 0 0
v3          ==> 7 7 7 7 7 7 7 7 7 7
v4          ==> 7 7 7 7 7 7 7 7 7 7
v5          ==> 3 4 5
```

STL vector에서 지원하는 연산:

v.size( );	vector에 살제 지정된 원소의 개수를 조사한다.
max_size	벡터가 관리할 수 있는 최대 요소 개수를 조사한다.
capacity( )	여유공간을 포함하여 물리적으로 실제 할당된 요소 개수를 구한다.
resize(n)	크기를 변경한다. 벡터의 원래 내용은 유지하며 새로 할당된 요소는 0으로 초기화된다.
reserve(n)	최소한의 크기를 지정하며 메모리를 미리 할당해 놓는다. 새로 할당된 요소는 초기화되지 않는다.
clear(n)	모든 요소를 삭제한다.
empty( )	제시한 vector가 완전히 비어있는지 조사한다.

기본 배열(array)가 아니라 vector를 사용할 때의 장점

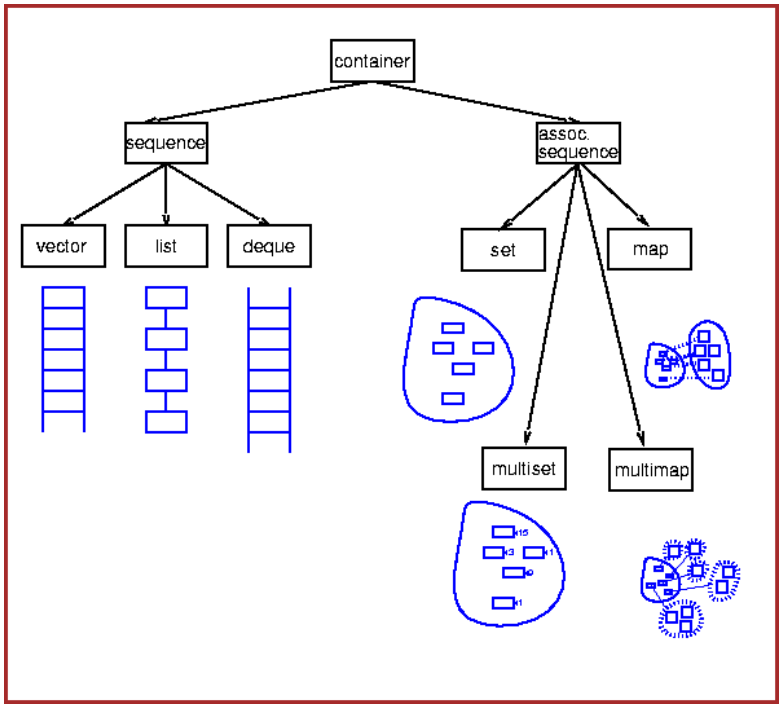
1) 더 이상 여러분이 OS에 memory call를 다루지 않아도 된다.

2) Pointer를 사용하지 않아도 된다.

3) 크게 제한에 대하여 더 이상 걱정할 필요가 없다.

4) 만일 오류가 있을 경우에는 표준화된 오류 메시지를 받을 수 있다.

5) 배열의 크기를 dynamic하게 조정할 수 있다. (컴파일 시간에 결정되지 않아도 된다.)



■ 지혜의 말씀 4-2: 프로그램 <코드>와 <변명>은 간결할수록 좋다.

4.9 STL에서 사용자가 부릴 수 있는 "재주"의 예

- 이미 구현된 STL 자료구조 중에서 선택할 수 있다.
- 자료구조란 어떤 member function( )을 사용하는 Class 이다.
- 동시에 해당 자료구조의 실제 구현 방법을 지시할 수 있다.
  예를 들어 stack을 vector로도 만들 수 있고 list로도 만들 수 있다.

- 자료구조는 동작에 대한 정의일 뿐, 그것을 실제 컴퓨터로 구현하는 것은 매우 다양한 방법이 있다.

- 어떤 특정한 위치에 상수시간(constant)에 갈 수 있고 그 앞 뒤 위치에 상수시간으로 갈 수 있으면 배열(vector)이다.

4.10 STL 사용 급수 (Levels of STL commanding techniques)

- STL은 게임이 아님은 알고 있다.
- STL이 컴파일러의 일종이 아닌 줄은 안다.
- STL이 C,C++에 끼워 쓰는 뭐 도구라는 이야기를 들어 본 것 같다.
- STL을 사용한 프로그램이나 코드를 확인할 수 있다.
- STL로 예제 프로그램을 짜 본 적이 있다.
- 어떤 문제를 받으면 STL의 어떤 기능을 사용할지 생각이 난다.

생각만 날 뿐이며 손은 따라가지 않는다.

- 어떤 문제를 받으면 STL로 된 코드가 떠오른다.
- STL의 기능을 조합해서 사용할 수 있다.

  예를 들어 연결 리스트의 원소를 Vector를 넣을 수 있다.

  vector의 원소에 list를 넣을 수 있다.
- 내가 만든 자료구조를 STL에 넣을 수 있다.
- STL의 기능 중에 성능의 차이를 알고 있다.
- STL 기본 자료구조를 고쳐서 복잡한 기능으로 추가 발전시킬 수 있다.
- STL로 만든 새로운 상위 단계의 data structure package를 만들 수 있다.
- 그걸 팔 수 있다.
- STL관련 20 시간짜리 강의를 대기업 연구소에 가서 할 수 있다.
- "달려라 STL!" 이라는 책을 쓸 수 있다.
- STL에 새로운 기능을 넣어서 표준화에 추가할 수 있다.

4.11 외장 기반 초대형 자료를 위한 STXXL: http://stxxl.org/

Standard Template Library for Extra Large Data Sets. 이 자료구조는 특별히 외장 장치기반의 자료구조를 위한 Template이다. 외장 장치에 저장된 자료구조와 메일 메모리에 저장된 자료의 성능 분석은 다르게 접근해야 한다. 외장 장치의 경우 단위 데이터를 CPU까지 읽어들이는데 시간은 길지만(이것은 일반적인 disc 기반을 가정), 한번에 큰 단위의 데이터를 읽을 수 있기 때문에 main memory에서 CPU(register)로 데이터 옮기는데 걸리는 시간과 직접적으로 비교할 수 없다. 가장 대표적인 자료구조로는 B-tree를 들 수 있다.

STXXL Layers의 특징

(1) The core of STXXL is an implementation of the C++ STL for external memory computations, STXXL implements containers and algorithms that can process huge volumes on disks.

(2) STXXL의 key features는 다음과 같다.

- Transparent support of parallel disks with parallel disk algorithms. STXXL supports the external memory algorithm library with parallel disks. The library is able to handle very large size up to dozens of terabytes. STXXL implementations benefit from overlapping of I/O and computation.

- A unique library feature called "pipelining" can save more than half the number of I/Os, by streaming data between algorithmic components.

- STL algorithms can be directly applied to STXXL containers. For internal computation, parallel algorithms from the MCSTL or **libstdc++** parallel mode are optionally utilized.

STXXL은 무료(free)이며 open source로서 쉽게 구할 수 있다. Boost Software License 1.0 라이센스 기반으로 활용할 수 있다.

참고 자료: **sdtc++.h**의 내용

```
// Copyright (C) 2003-2013 Free Software Foundation, Inc.
/** @file stdc++.h
 *  This is an implementation file for a precompiled header.
 */
```

```
// C

#ifndef _GLIBCXX_NO_ASSERT
#include <cassert>
#endif
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <ciso646>
#include <climits>
#include <clocale>
#include <cmath>
#include <csetjmp>
#include <csignal>
#include <cstdarg>
#include <cstddef>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>

#if __cplusplus >= 201103L
#include <ccomplex>
#include <cfenv>
#include <cinttypes>
#include <cstdalign>
#include <cstdbool>
#include <cstdint>
#include <ctgmath>
#include <cwchar>
#include <cwctype>
#endif
```

```
// C++
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <new>
#include <numeric>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>
```