|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | |  | | **제11회 공개SW 개발자 대회 결과 보고서** | |  | |

**□ 참가팀 개요**

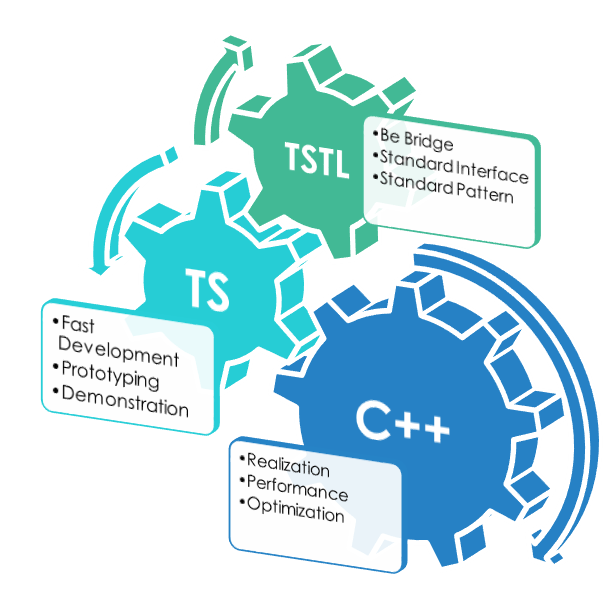
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **구 분** | | **세부내용** | | | |
| **팀 명** | | TSTL | | | **총 인원 ( 1 명 )** |
| **팀 구 성** | | **성명** | **소속** | **부서/학과** | **직위/학년** |
| **팀 장** | | 남정호 | 아이지넷 | 개발실 | 대리급 |
| **참가**  **지원** | **부문** |  | |  | |
| **분야** |  |  |  |  |
|  |  |  |  |
|  | |  | | | |

**□ 출품작 결과 보고서**

|  |  |
| --- | --- |
| **출품작명** | TSTL (TypeScript-STL) |
| **프로그램 등록URL** | <https://github.com/samchon/tstl> |
| **출품작 소개**  **(요약)** | **TypeScript** 의 **STL** (Standard Template Library) 구현체   * Containers: 컬렉션 프레임워크 * Iterators: 이터레이터 패턴 * Algorithms: 알고리즘 함수들 * Functors: 유틸리티 객체들   TSTL (TypeScript-STL) 은, C++ 표준화 위원회 (C++ Standard Committee) 가 정의한, 표준 템플릿 라이브러리 (Standard Template Library) 를 TypeScript 에 구현한 오픈소스 프로젝트입니다.  무분별한 하드코딩과 무질서한 자료구조 사용에 지친 당신, 오늘부터 TSTL 하십시오. C++ 계열 언어들만의 전유물이었던 체계화된 자료구조, 제네릭 & 함수형 프로그래밍, 이젠 웹 (또는 NodeJS) 에서도 가능합니다. |

**1. 개발배경 및 목적**

저는 알고리즘 개발자입니다. 저는 알고리즘을 구현할 때, Script 언어로 빠르게 Prototype 을 만들어보고, 차후 C++ 로 최적화하는 방법을 선호합니다. 아래는 저의 단골 개발 프로세스 (스타일) 입니다.

* **Fast Development** by JavaScript
  + Prototyping
  + Simulation in data level
* **Proof** & Visualize by HTML
  + Production
  + Demonstration with visualization
  + High-level Simulation
* **Optimization** by Migration to C++
  + Migration
    - Consider efficiency
    - Partial or Full migration
  + Integration with TS/HTML

그리고 이렇게 알고리즘을 개발하면서, 느꼈던 두 가지의 불편사항들로 인해 **TSTL**을 개발하게 됩니다.

**1.1. JS 는 기저 라이브러리가 너무 부실하더라**

JS 는 기저 라이브러리가 너무나도 부실합니다. 제공되는 Containers 는 달랑 Array 와 Object (dynamic object) 뿐이며, 알고리즘을 구현하기 위한 기본 제공 함수들도 너무 적습니다. 또한, 유틸리티 객체들도 너무나도 단조로우며 알고리즘의 경우와 마찬가지로 기본으로 제공되는 features 가 너무 적습니다.

**STL**을 **TypeScript**에 구현해 놓으면 JS 에서도, 다채로운 라이브러리를 사용할 수 있게 됩니다. 단순 라이브러리도 아니고 C++ 표준화 위원회가 십 수년에 걸쳐 고민하고 또 고민하여 정립한 표준 라이브러리입니다.

* Containers & Iterators
* Algorithms
* Functors

**1.2. Migration 이 너무 번겁더라**

저는 앞서 언급한 바와 같이, Script 언어 (JS) 로 빠르게 Prototype 을 만들어보고, 차후 C++ 로 최적화하는 방법을 선호합니다.

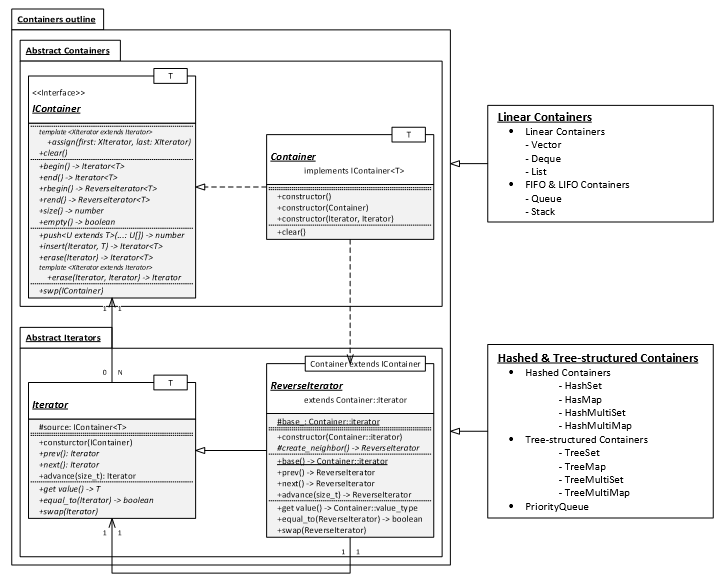
하지만, Prototyping 을 끝내고, JS로 작성한 코드를 C++ 로 마이그레이션을 하는 과정은 언제나 지루하고 고단합니다. 특히 JS와 C++ 은 기저 라이브러리 레벨에서 인터페이스가 서로 상이하니, 이게 더더욱 불편하고 고단한 작업이 아니었나 싶습니다.

만일 JS와 C++ 이 기저 라이브러리 수준에서 인터페이스가 같다면, 동일한 자료구조와 동일한 패턴을 지닌다면 얼마나 편리할까? 그런 생각에서 제작하게 되었습니다; **TSTL**.

**2. 개발환경 및 개발언어**

* 개발환경
  + 설계 도구: Visio
  + 코딩 도구: Visual Studio Code
  + 배포 장소: GitHub, NPM Modules
    - <https://github.com/samchon/tstl>
    - <https://npmjs.com/tstl>
* 개발언어: TypeScript
* 매뉴얼
  + Class Diagram: <http://samchon.github.io/tstl/design/class_diagram.pdf>
  + API Documents: <http://samchon.github.io/tstl/api/>
  + Guide Documents: <https://github.com/samchon/tstl/wiki>

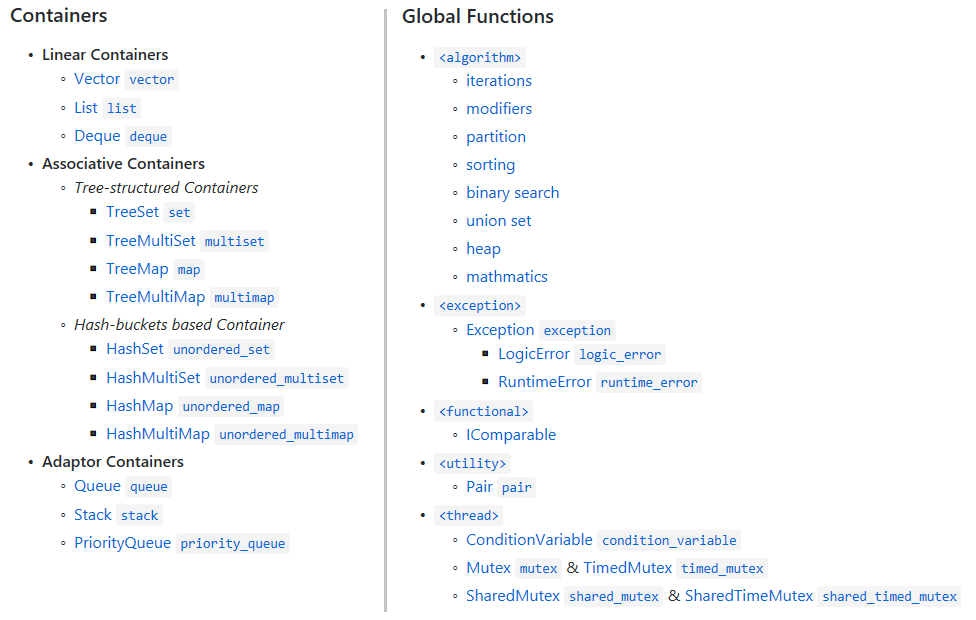
**3. 시스템 구성 및 아키텍처**



TSTL 의 모듈 구성은 아래와 같습니다.

* **Containers** & **Iterators**
  + C++ 의 Collection Framework
  + 다양한 Containers 와
  + 정규화된 Iterator Pattern 을 제공함
* **Algorithms**
  + 알고리즘 함수들
  + sorting, mathematics, partitions 등
  + Iterator Pattern + Functional Programming
* **Functors**
  + 기타 유틸리티 객체들

**4. 프로젝트 주요 기능**



**4.1. Containers**

Java 의 Collection Framework 격에 해당하는 모듈로써, 다채로운 Containers 와 정규화된 Iterator Pattern 을 제공합니다. 개인적으로 STL 의 Containers 가 Java Collection Framework 보다 다음 세 가지 관점에서 더 우수하다고 생각합니다.

* 일관성
* 적절성
* 확장성

현재 TSTL 에서 제공되는 컨테이너는 크게 세 가지 (Linear, Adaptor & Associative) 카테고리로 나눌 수 있으며, 그 것들의 종류는 아래와 같습니다.

* **Linear Containers**
  + vector
  + list
  + deque
* **Adaptor Containers**
  + stack
  + queue
  + priority\_queue
* **Associative Containers**
  + Atomic Containers
    - set Set
    - multiset TreeMultiSet
    - unordered\_set HashSet
    - unordered\_multiset HashMultiSet
  + Dictionaries
    - map TreeMap
    - multimap TreeMultiMap
    - unordered\_map HashMap
    - unordered\_multimap HashMultiMap

**4.2. Algorithms**

다양한 알고리즘 함수들의 모음입니다.

STL 의 알고리즘 함수들은 일관된 패턴을 지니며 (일관적이며), 용이성과 확장성이 매우 우수합니다. 또한, STL 에서 제공되는 알고리즘 함수들은 크게 다음 두 가지 특성을 지닙니다.

* Iterator Pattern
* Functional Programming

제공되는 알고리즘 함수들의 카테고리는 아래와 같습니다.

* iterations
* modifiers
* sortings
* heaps
* binary\_searches
* partitions
* union\_sets
* mathmatics

**4.3. Functors**

다양한 유틸리티성 객체들의 모음으로써, 다르게 표현하면 기타 항목이라 할 수 있겠습니다.

제공되는 Feature들은 아래와 같습니다.

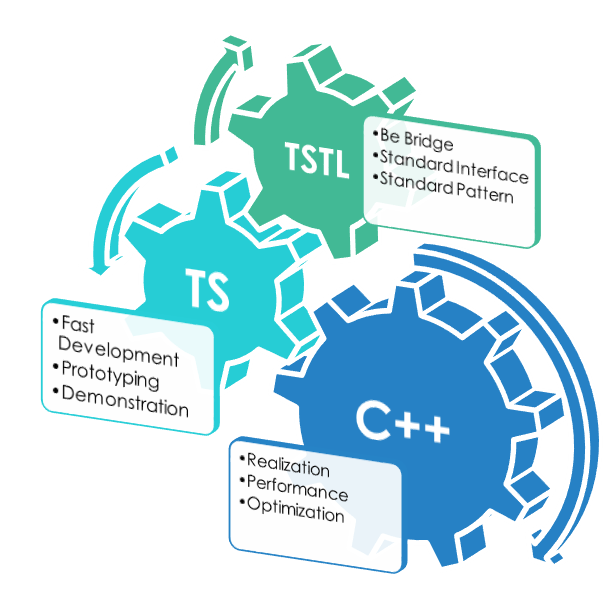
* exceptions
  + 예외사항 객체들
  + JS 에는 예외사항을 발생시키는 객체가 Error 타입 뿐이니, 이를 보완코자 구현함
* functional
  + binary functions
  + hash functions
* utility
* (fake) thread
  + Asynchronous Critical Section Objects
    - 비동기 이벤트간 임계영역 제어
    - JS 나름의 스레드를 흉내내어 구현할 수 있다.
  + 제공되는 클래스들은 다음과 같다.
    - condition\_variable
    - mutex
    - timed\_mutex
    - shared\_mutex
    - shared\_timed\_mutex
    - experiments::semaphore

STL이 개정될 때마다 이 Functors 모듈의 규모가 기하급수적으로 커지는 경향이 있지만, TSTL 역시 STL의 개정에 발맞춰, 신규 기능들을 계속 추가해 나갈 것입니다. 현재 차후 STL 개정을 통해 추가될 것으로 예상되는 기능들은 아래와 같습니다.

* File System (17)
* Standard UI Library (20)
* Standard Network Library (20)

**5. 기대효과 및 활용분야**

**5.1. 기대효과**



**1) JavaScript 에 풍부한 라이브러리 제공**

JS 는 기저 라이브러리가 너무나도 부실합니다. 제공되는 Containers 는 달랑 Array 와 Object (dynamic object) 뿐이며, 알고리즘을 구현하기 위한 기본 제공 함수들도 너무 적습니다. 또한, 유틸리티 객체들도 너무나도 단조로우며 알고리즘의 경우와 마찬가지로 기본으로 제공되는 features 가 너무 적습니다.

TSTL 을 사용하면 JS 에서도, 다채로운 라이브러리를 사용할 수 있습니다. 단순 라이브러리도 아니고 C++ 표준화 위원회가 십 수년에 걸쳐 고민하고 또 고민하여 정립한 표준 라이브러리입니다. 일관성과 적절성, 그리고 확장성까지 두루두루 갖춘 표준 템플릿 라이브러리를 JS 에서 사용하십시오.

* Containers & Iterators
* Algorithms
* Functors

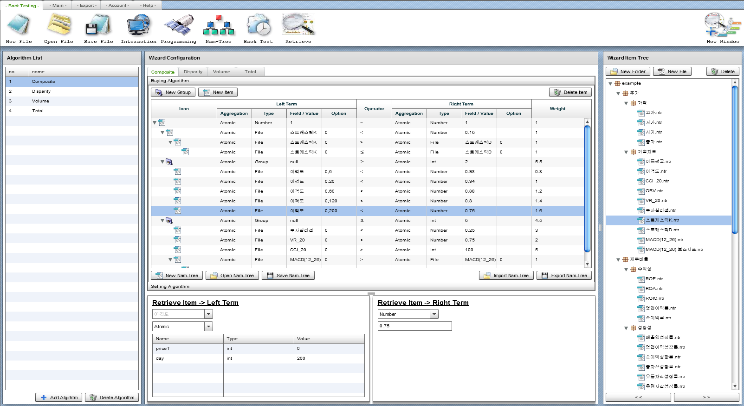
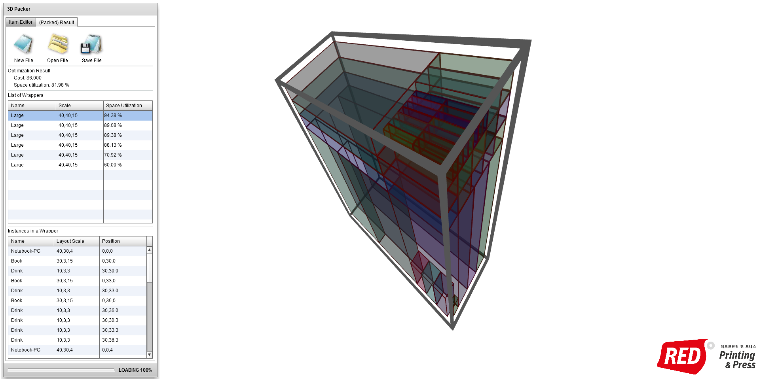
**2) C++ 과의 연동 효율 증대**

TSTL 을 사용함으로써, JS 와 C++ 은 기저 수준에서 같은 인터페이스를 지니게 됩니다.

이제부터 JS 와 C++ 은 STL의 이름 아래에 동일한 자료구조, 동일한 인터페이스의 컨테이너와 라이브러리 함수들을 지니게 됩니다. TS 🡪 C++ 로 마이그레이션을 행하던, 혹은 TS와 C++ 시스템을 각기 연동하던, 여러분은 설계 수준에 보다 더 집중할 수 있게 됩니다.

**5.2. 활용사례**

TSTL은 지난 3년간 다양한 상용 프로젝트들에 적용하면서 안정화를 시키고, 꾸준히 개선해 왔습니다. 때문에 TSTL의 품질에 대해서는 매우 자신 있고, 스스로도 자부심을 느끼고 있습니다. TSTL. 안심하고 쓰십시오.

* **Nam-Tree**
  + 간이 A.I. 모형
  + 고차원적인 라이브러리의 필요
  + Web 시각화 & C++ Migration
* **Samchon Simulation**
  + 주식 시뮬레이션 & 시스템 트레이딩
  + 고차원적인 라이브러리의 필요
  + Web 시각화 & C++ Integration
* **Samchon Framework**
  + OON (Object Oriented Network) Framework
  + Integration with C++
* **3D Bin Packing**
  + 공간배치 최적화
  + Web 을 통한 시각화 (3D Modeling)
  + C++ Migration
* **Folding**
  + 접지 단가 최적화
  + Containers & Algorithms 의 필요
* **Button Finder**
  + 이미지 검색기
  + Integration with C++
    - 프론트: JS + HTML
    - 분산처리: NodeJS
    - 이미지 프로세싱: C++
* **Kiosk Editor**
  + Cloud Project
  + 고차원의 Container 와
  + Asynchronous Critical Section 필요
* **Auto HL**
  + Cloud 장비관리
  + Kiosk Editor 와 동일한 이유
* **Package Retriever**
  + 보험 검색엔진
  + Collection Framework 와 Algorithms 및
  + Asynchronous Critical Section 의 필요

