The Rust Programming Language

쉽고 안전한 시스템 프로그래밍

김지현

2015-05-09

발표자 소개

13학번 김지현

UPnL 서버관리자 (2014~now)

@simnalamburt

Hyeon Kim
simnalamburt

Seoul, South Korea
simnalamburt@gmail.com
http://hyeon.me
Joined on 15 May 2013

37 Followers **353** Starred **93** Following

Organizations

Contributions | Repositories | Public activity

Popular repositories

- **vim-mundo** — Visualize your Vim undo tree — 14 ★
- **snucse** — Happy Campus Life — 6 ★
- **dotfiles** — dotfiles — 5 ★
- **gulp-slm** — Compile Slm templates with gulp — 5 ★
- **matrix** — Gives you matrix-like feeling — 4 ★

Repositories contributed to

- swsnu/**SNUCMS**
- hifive-snu/**hifive-test-explo** — hifive test explorer
- BradRuderman/**meteor_ha**
- wafflestudio/**snutt** — SNU Timetable
- sgkim126/**snucse.spline**

Contributions

May Jun Jul Aug Sep Oct Nov Dec Jan Feb

M
W
F

Summary of Pull Requests, issues opened, and commits. Learn more.

Contributions in the last year: 1,868 total (May 8, 2014 – May 8, 2015)
Longest streak: 26 days (March 15 – April 9)

Contribution activity

89 commits

Pushed 2 commits to bacchus-snu/bacchus-snu.github.io May 7

Pushed 3 commits to simnalamburt/snucse May 6 – May 7

Pushed 22 commits to simnalamburt/snucse.k-means May 6 – May 7

Pushed 19 commits to simnalamburt/snucse.matmul May 6

Pushed 4 commits to simnalamburt/obj-rs May 5 – May 6

Pushed 1 commit to simnalamburt/fate May 6

target	시스템 프로그래밍을 하고싶으신분, 하게되실분
non-target	시스템 프로그래밍 안할 사람

이 발표는 새내기들을 위해
난이도조절 되어있습니다.

Why Rust

<****> 러스트를 시작하고 인기남이 되었어요!

Fast	빠르다
Safe	안전하다
Easy	쉽다

Fast	빠르다
Safe	안전하다
Easy	쉽다


Dennis MacAlistair Ritchie
1941 ~ 2011
UNIX, C
THE JAPAN PRIZE FOUNDATION

malloc(), free()

- 원하는 양의 메모리를 할당, 프로그래머에게 준다

```
void *malloc(size_t size);
void free(void *ptr)
```

- 프로그래머에겐 정리의 의무가 있음

malloc(), free()

- 받아왔던 주소값을 free()에 넘기면 끗

```
void *mem = malloc(100);

/* ... */

free(mem);
```

- 근데 까먹고 안하면 Leakage

malloc(), free()

- 중간에 함수 조기종결이라도 한다면?

```
void *mem = malloc(100);
/* ... */

if (some_problem) {
  free(mem);
  return;
}
/* ... */
free(mem);
```

- 꼼꼼해져야함

malloc(), free()

- 할당을 셋 이상 하면?
- 조기종결을 세번 이상 하면?

malloc(), free()

- 할당을 셋 이상 하면?
- 조기종결을 세번 이상 하면?

```c
void *mem1 = malloc(100);
void *mem2 = malloc(1000);
/* ... */

if (some_problem) {
  free(mem1);
  free(mem2);
  return;
}
/* ... */
void *mem3 = malloc(10000);
if (another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}
/* ... */
if (yet_another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}

/* ... */
free(mem1);
free(mem2);
free(mem3);
```

```c
void *mem1 = malloc(100);
void *mem2 = malloc(1000);
/* ... */

if (some_problem) {
  free(mem1);
  free(mem2);
  return;
}
/* ... */
void *mem3 = malloc(10000);
if (another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}
/* ... */
if (yet_another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}

/* ... */
free(mem1);
free(mem2);
free(mem3);
```

여기서 뭔가 실수로 안썼다면?

```c
void *mem1 = malloc(100);
void *mem2 = malloc(1000);
/* ... */

if (some_problem) {
  free(mem1);
  free(mem2);
  return;
}
/* ... */
void *mem3 = malloc(10000);
if (another_problem) {
  free(mem1);
  free(mem2);

  return;
}
/* ... */
if (yet_another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}

/* ... */
free(mem1);
free(mem2);
free(mem3);
```

여기서 뭔가 실수로 안썼다면?

```c
void *mem1 = malloc(100);
void *mem2 = malloc(1000);
/* ... */

if (some_problem) {
  free(mem1);
  free(mem2);
  return;
}
/* ... */
void *mem3 = malloc(10000);
if (another_problem) {
  free(mem1);
  free(mem2);

  return;
}
/* ... */
if (yet_another_problem) {
  free(mem1);
  free(mem2);
  free(mem3);
  return;
}

/* ... */
free(mem1);
free(mem2);
free(mem3);
```

여기서 뭔가 실수로 안썼다면?

사실 이렇게 하면 됨

```
void *mem1 = malloc(100);
void *mem2 = 0;
void *mem3 = 0;
/* ... */

if (some_problem) { goto ERROR; }

mem2 = malloc(1000);
/* ... */
mem3 = malloc(10000);

if (another_problem) { goto ERROR; }

/* ... */

if (yet_another_problem) { goto ERROR; }

/* ... */
ERROR:
free(mem1); mem1 = 0;
free(mem2); mem2 = 0;
free(mem3); mem3 = 0;
```

- 문제 해결?

사실 이렇게 하면 됨

```
void *mem1 = malloc(100);
void *mem2 = 0;
void *mem3 = 0;
/* ... */

if (some_problem) { goto ERROR; }

mem2 = malloc(1000);
/* ... */
mem3 = malloc(10000);

if (another_problem) { goto ERROR; }

/* ... */

if (yet_another_problem) { goto ERROR; }

/* ... */
ERROR:
free(mem1); mem1 = 0;
free(mem2); mem2 = 0;
free(mem3); mem3 = 0;
```

- 문제 해결?

- malloc(), free() 외에 다른 리소스 할당을 스택 쌓듯이 하면?
- double-free, use-after-free 에서 안전한가?
- 여전히 free 까먹고 안할 가능성

사실 이렇게 하면 됨

```
void *mem1 = malloc(100);
void *mem2 = 0;
void *mem3 = 0;
/* ... */

if (some_problem) { goto ERROR; }

mem2 = malloc(1000);
/* ... */
mem3 = malloc(10000);

if (another_problem) { goto ERROR; }

/* ... */

if (yet_another_problem) { goto ERROR; }

/* ... */
ERROR:
free(mem1); mem1 = 0;
free(mem2); mem2 = 0;
free(mem3); mem3 = 0;
```

- 문제 해결?

- malloc(), free() 외에 다른 리소스 할당을 스택 쌓듯이 하면?
- double-free, use-after-free 에서 안전한가?
- 여전히 free 까먹고 안할 가능성

사실 이렇게 하면 됨

```
void *mem1 = malloc(100);
void *mem2 = 0;
void *mem3 = 0;
/* ... */

if (some_problem) { goto ERROR; }

mem2 = malloc(1000);
/* ... */
mem3 = malloc(10000);
/* ... */

if (yet_another_problem) { goto ERROR; }

/* ... */
ERROR:
free(mem1); mem1 = 0;
free(mem2); mem2 = 0;
free(mem3); mem3 = 0;
```

- 문제 해결?
- malloc(), free() 외에 다른 리소스 할당은 스택 쌓듯이 하면?
- double-free, use-after-free에 대한 안전한가?
- 여전히 free 까먹고 안할 가능성

자동으로 하고싶다!


C++
Golden Rule
Resource Acquisition Is Initialization

C++, RAII

- 소멸자 문법

```
struct raii {
  raii() { hello(); }
  ~raii() { bye(); }
};

int main() {
  for (int a = 0; a < 10; ++a) {
    raii a; // 자동으로 hello() 가 호출됨

    /* ... */

  } // 스코프를 빠져나가면 자동으로 bye() 가 호출됨
}
```

- 괄호가 열리면, 닫힐것이 무조건 보장됨

스마트 포인터

- 똑똑한 포인터

```
{
  int *dumb = new int(10);
  /* ... */

} // <- LEAK!

{
  auto smart = unique_ptr(new int(10));
  /* ... */

} // <- RAII ♥
```

이제 진짜 된건가?!


NOPE

C++의 문제

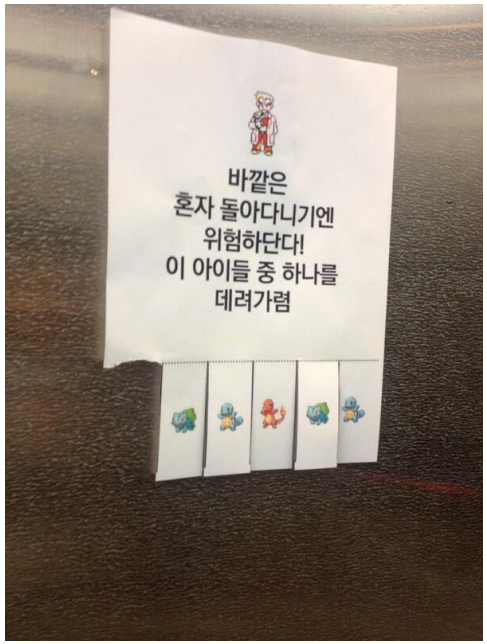
1. 배우기 어렵다
2. 실수하기 쉽다

C++의 문제

1. 배우기 어렵다
2. 실수하기 쉽다

C++의 문제

1. 배우기 어렵다
2. 실수하기 쉽다



C++ is not even close to memory safety

```
auto list = vector<int>();

/* ... */

auto& elem = list[0];
list.push_back(100);
cout << elem; // BOOM!!
```

- Dangling reference

Case study: Firefox

- 파이퍼폭스 버그, 보안취약점들의 원인중 double free, user after free가 여전히 상위권임

Back to search

MFSA2014-30 Firefox: Use-after-free in TypeObject (CVE-2014-1512)

Severity	CVSS	Published	Added	Modified
9	(AV:N/AC:M/Au:N/C:C/I:C/A:C)	March 17, 2014	March 18, 2014	July 20, 2014

Description

Use-after-free vulnerability in the TypeObject class in the JavaScript engine in Mozilla Firefox before 28.0, Firefox ESR 24.x before 24.4, Thunderbird before 24.4, and SeaMonkey before 2.25 allows remote attackers to execute arbitrary code by triggering extensive memory consumption while garbage collection is occurring, as demonstrated by improper handling of BumpChunk objects.

Back to search

MFSA2015-18 Firefox: Double-free when using non-default memory allocators with a zero-length XHR (CVE-2015-0828)

Severity	CVSS	Published	Added	Modified
7	(AV:N/AC:M/Au:N/C:P/I:P/A:P)	February 24, 2015	February 25, 2015	February 25, 2015

Description

Double free vulnerability in the nsXMLHttpRequest::GetResponse function in Mozilla Firefox before 36.0, when a nonstandard memory allocator is used, allows remote attackers to execute arbitrary code or cause a denial of service (heap memory corruption) via crafted JavaScript code that makes an XMLHttpRequest call with zero bytes of data.

Case study: Mabinogi 2

상황

```
// 결과를 보존. 사실은 마우스를 클릭하여 실제로 픽킹한 것은 아니고
// 호버링만 하고 있는 상태라는 사실을 주의할 것. 피킹은 다른 곳에서
// 나중에 처리한다.
PickingResultProcedure::UpdateHovering(sim->GameContext(), picking, skewered);
```

여기서 크래시 0x00000010 읽다가 사망
재현조건: 던전을 플레이한다 (;;;)

Case study: Mabinogi 2

상황

```
// 결과를 보존. 사실은 마우스를 클릭하여 실제로 픽킹한 것은 아니고
// 호버링만 하고 있는 상태라는 사실을 주의할 것. 피킹은 다른 곳에서
// 나중에 처리한다.
PickingResultProcedure::UpdateHovering(sim->GameContext(), picking, skewered);
```

여기서 크래시 0x00000010 읽다가 사망
재현조건: 던전을 플레이한다 (;;;)

Admit it

C++ is not that good

Admit it

C++ is not that good

여전히 좋다고 생각하신다면 스톡홀름 증후군을 의심해보십시오

“원래 기존의것을 낫게 만드는게
새 언어 만들어서 하는것보다 훨씬 힘들어.”

멀티코어 컴퓨팅 연구실
이재진 교수님

5월 6일 301동 101호
멀티코어 수업 시간중 발췌

"원래 기존의것을 낫게 만드는게
새 언어 만들어서 하는것보다 훨씬 힘들어."

멀티코어 컴퓨팅 연구실
이재진 교수님

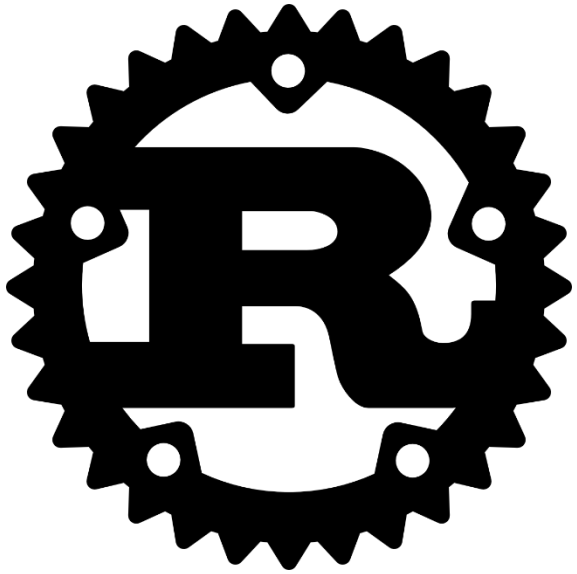
5월 6일 301동 101호
멀티코어 수업 시간중 발췌

"기존의것을 낫게 만드느니
새 언어 만드는게 훨씬 쉬워."


그런데
그것이
실제로 일어났습니다. ㅋㅋ
ㅋㅋ
ㅋㅋ

mozilla


“To design and implement a safe, concurrent, practical, static systems language.”
mozilla



http://rust-lang.org

```
fn main() {
    println!("Hello, world!");
}
```

Rust is a systems programming language that runs blazingly fast, prevents almost all crashes*, and eliminates data races.

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Why Rust

<****> 러스트를 시작하고 인기남이 되었어요!

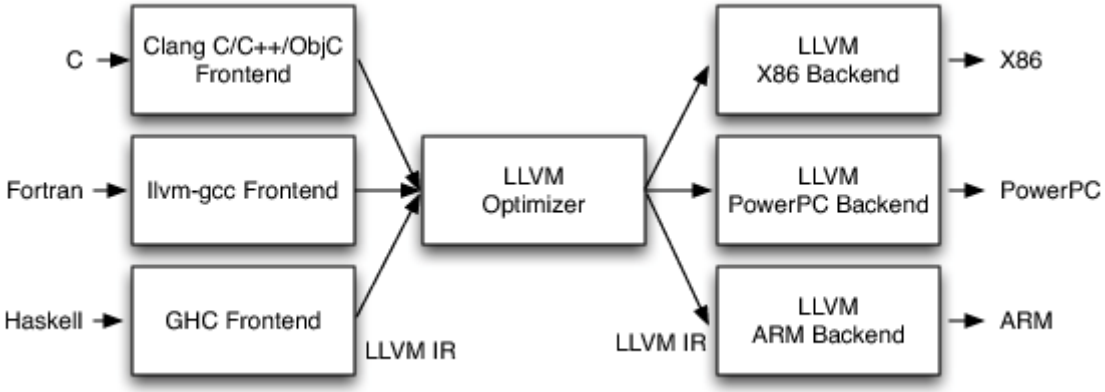
Fast	빠르다
Safe	안전하다
Easy	쉽다

Fast	빠르다
Safe	안전하다
Easy	쉽다

LLVM

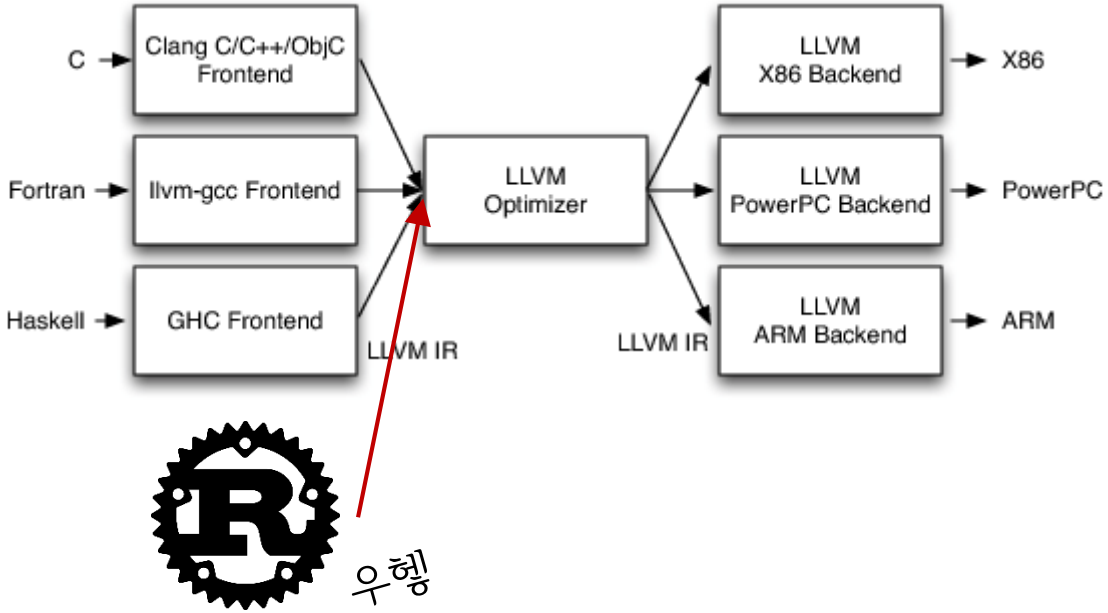
LLVM

- 컴파일러 기반구조
- 훌륭하신 분들이 잘 만들어놨음



LLVM

- 컴파일러 기반구조
- 훌륭하신 분들이 잘 만들어놨음




Fast | 빠르다
Safe | 안전하다
Easy | 쉽다

Recommended Version:
1.0.0-beta.4 (Windows installer)

Install

Other Downloads

Rust is a systems programming language that runs blazingly fast, prevents almost all crashes*, and eliminates data races.

Show me more!

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Run

```
// This code is editable and runnable!
fn main() {
    // A simple integer calculator:
    // `+` or `-` means add or subtract by 1
    // `*` or `/` means multiply or divide by 2

    let program = "+ + * - /";
    let mut accumulator = 0;

    for token in program.chars() {
        match token {
            '+' => accumulator += 1,
            '-' => accumulator -= 1,
            '*' => accumulator *= 2,
            '/' => accumulator /= 2,
            _ => { /* ignore everything else */ }
        }
    }

    println!("The program \"{}\" calculates the value {}",
             program, accumulator);
}
```

More examples

Book
Reference

Book
Reference

ocs
ocs

GitHub
User Forum
IRC
Reddit

Stack Overflow
Twitter
Blog
Calendar

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

ge
t all

Recommended Version:
1.0.0-beta.4 (Windows installer)

Install

Other Downloads

```
ole and runnable!

er calculator:
ns add or subtract by 1
ns multiply or divide by 2

+ * - /";
or = 0;

ram.chars() {

cumulator += 1,
cumulator -= 1,
cumulator *= 2,
cumulator /= 2,
 ignore everything else */ }


gram \"{}\" calculates the value {}",
program, accumulator);
}
```

Run

More examples

Book Reference | Book Reference | GitHub User Forum IRC Reddit | Stack Overflow Twitter Blog Calendar

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Recommended Version:
1.0.0-beta.4 (Windows installer)

Install

Other Downloads

```
ble and runnable!

er calculator:
s add or subtract by 1
s multiply or divide by 2

+ * - /";
or = 0;

ram.chars() {

cumulator += 1,
cumulator -= 1,
cumulator *= 2,
cumulator /= 2,
 ignore everything else */ }


gram \"{}\" calculates the value {}",
 accumulator);
}
```

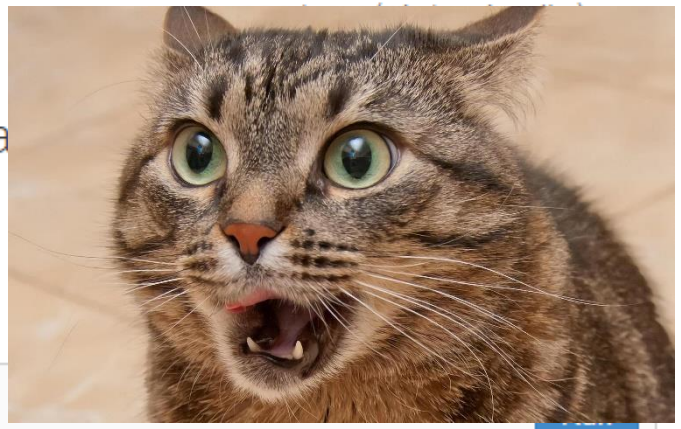
Run

More examples

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Recommended Version:



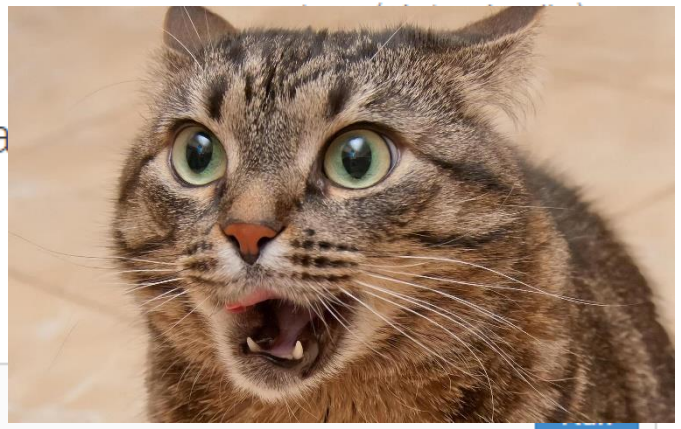
```
er calculator:
ns add or subtract by 1
ns multiply or divide by 2

+ * - /";
or = 0;

ram.chars() {

cumulator += 1,
cumulator -= 1,
cumulator *= 2,
cumulator /= 2,
 ignore everything else */ }


gram \"{}\" calculates the value {}",
, accumulator);
}
```

More examples

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inferenc
- minimal runt
- efficient C bi

Recommended Version:




```
er calculator:
s add or subtract by 1
s multiply or divide by 2
```

```
/ }
e value {}",
}
```

More examples

Fast	빠르다
Safe	안전하다
Easy	쉽다

쉬웡

C++

- 암걸리는 문법
- 배울게 걍 많음
- Makefile, autoconf, cmake, ninja, ...
- 통일된 패키지매니저 X

Rust

- 젊음
- 간단함
- cargo

- cargo https://crates.io
- 매크로, annotation, ...

디펜던시 관리

- 걍 pip, npm, gem 얘네랑 다를거 없음

```
[dependencies]
time = "*"
glutin = "*"
```

- 이러면 원터치로 다 설치됨

- 반면 C++에선 컴파일옵션 바꾸고 경로설정하고 라이브러리마다 설치방법 다 다르고 난리를 치지

rustdoc 기본 내장

- 코드에 주석만 적절히 달면

```
76  /// Conversion from `RawObj`'s raw data.
77  pub trait FromRawVertex {
78      /// Build vertex and index buffer from raw object data.
79      fn process(vertices: Vec<f32x4>, normals: Vec<f32x4>, polygons: Vec<Polygon>) -> ObjResult<(Vec<Self>, Vec<u16>)>;
80  }
81
82  /// Vertex data type of `Obj` which contains position and normal data of a vertex.
83  #[derive(Copy, PartialEq, Clone, Debug)]
84  pub struct Vertex {
85      /// Position vector of a vertex.
86      pub position: [f32; 3],
87      /// Normal vertor of a vertex.
88      pub normal: [f32; 3],
89  }
90
91  #[cfg(feature = "glium-support")]
92  implement_vertex!(Vertex, position, normal);
93
94  impl FromRawVertex for Vertex {
95      fn process(positions: Vec<f32x4>, normals: Vec<f32x4>, polygons: Vec<Polygon>) -> ObjResult<(Vec<Self>, Vec<u16>)> {
96          let mut vb = Vec::with_capacity(polygons.len() * 3);
97          let mut ib = Vec::with_capacity(polygons.len() * 3);
98          {
99              let mut cache = HashMap::new();
```

rustdoc 기본 내장

Crates

obj

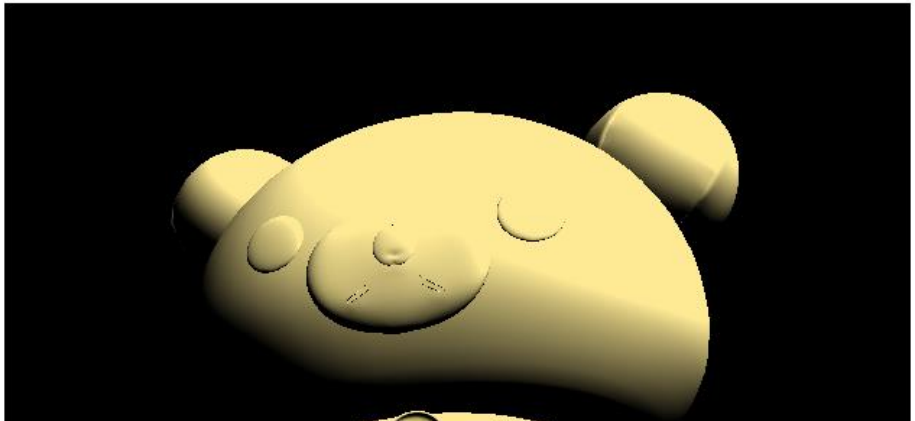
Click or press 'S' to search, '?' for more options...

Crate obj

[stability] [-] [+] [src]

[-] Wavefront OBJ parser for Rust. It handles both `.obj` and `.mtl` formats. GitHub

```
use std::fs::File;
use std::io::BufReader;
use obj::*;

let input = BufReader::new(File::open("tests/fixtures/normal-cone.obj").unwrap());
let dome: Obj = load_obj(input).unwrap();

// Do whatever you want
dome.vertices;
dome.indices;
```



rustdoc 기본 내장

Modules

raw	Provides low-level API for Wavefront OBJ format.

Structs §

Obj	3D model object loaded from wavefront OBJ.
Position	Vertex data type of `Obj` which contains only position data of a vertex.
Vertex	Vertex data type of `Obj` which contains position and normal data of a vertex.

Traits

FromRawVertex	Conversion from `RawObj`'s raw data.

Functions

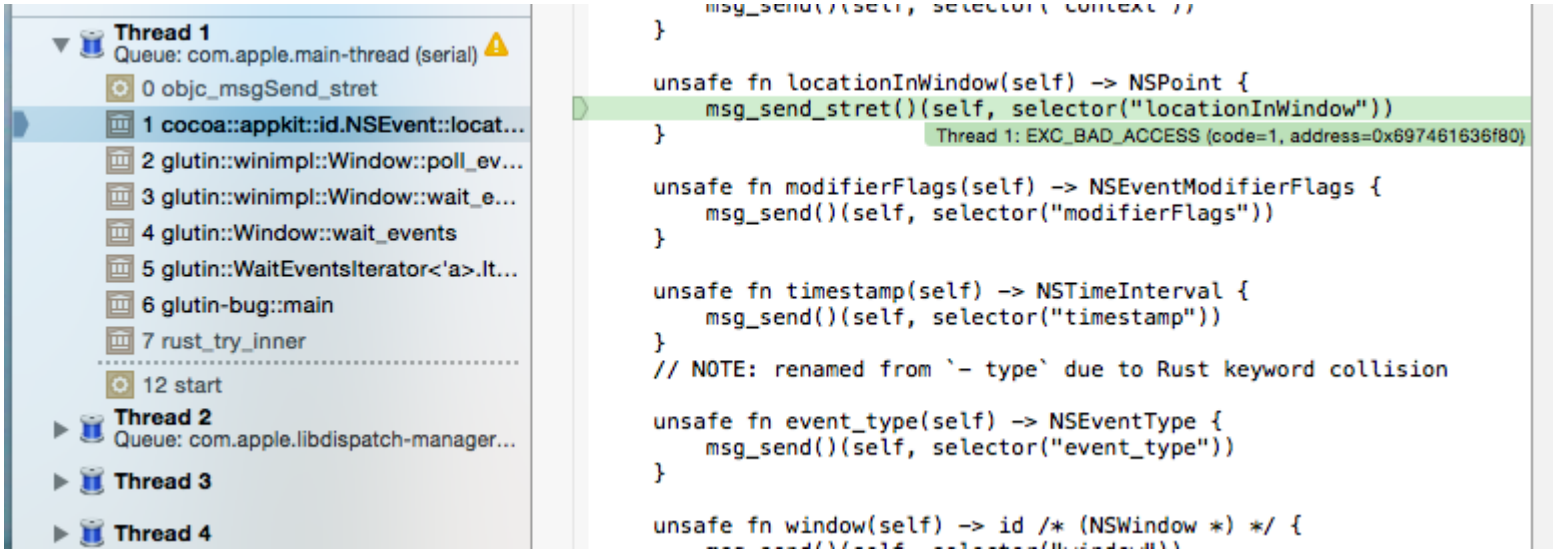
load_obj	Load a wavefront OBJ file into Rust & OpenGL friendly format.

Type Definitions

ObjResult	A type for results generated by `load_obj` and `load_mtl` where the `Err` type is hard-wired to `ObjError`

디버깅

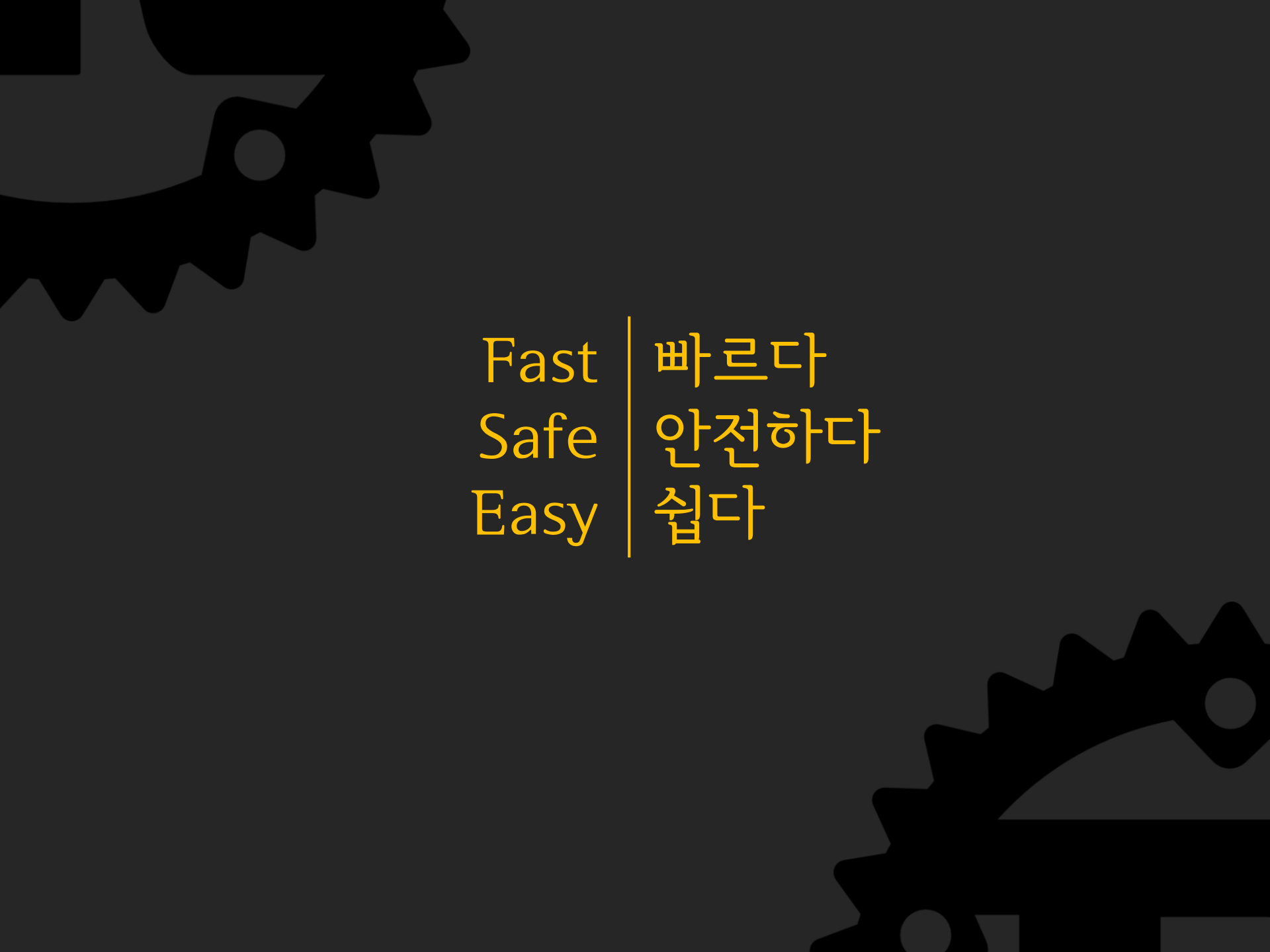
- gdb, lldb, 자기가 원하는거 쓰면 됨

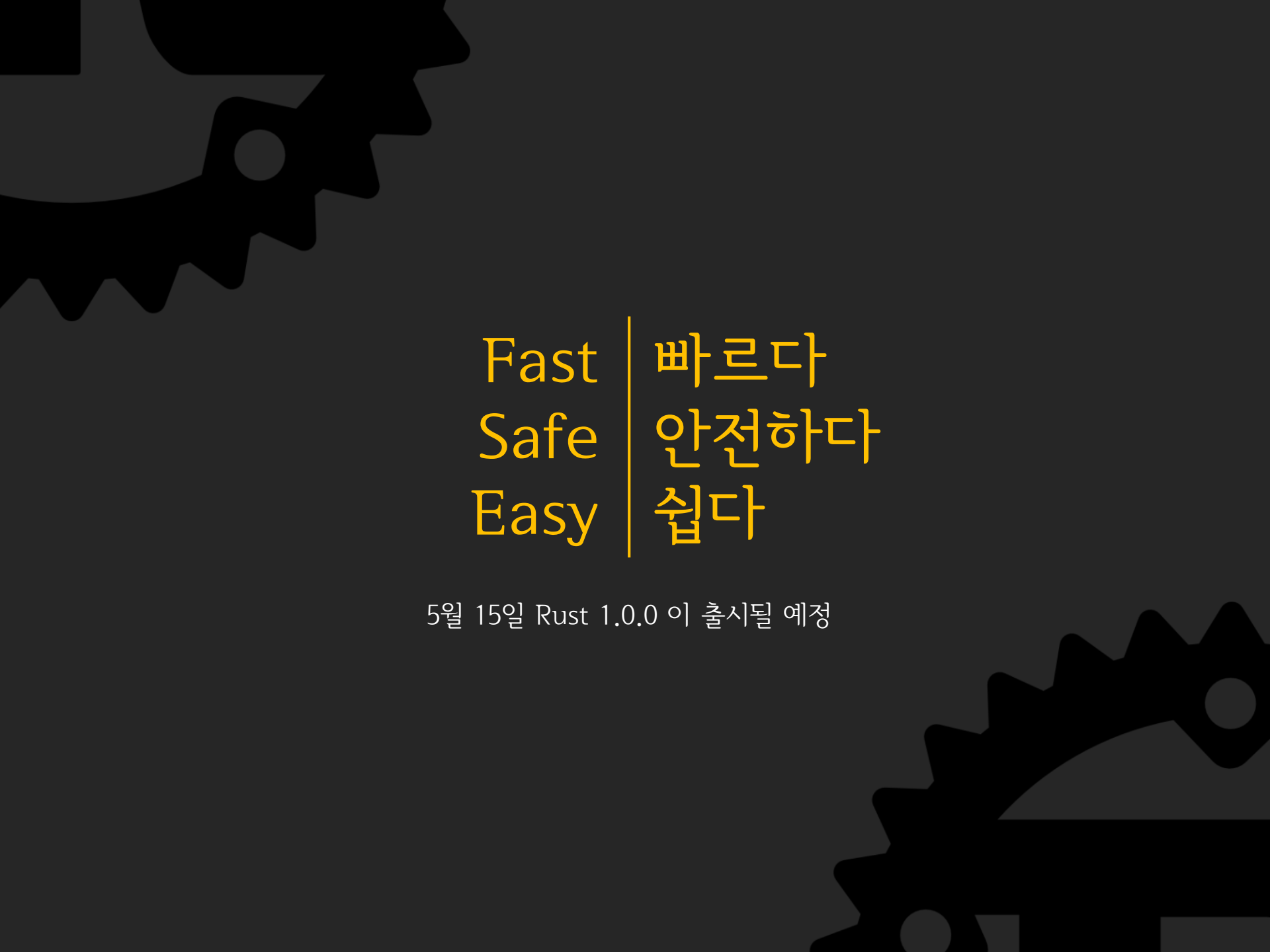


- LLVM 기반이다보니 Xcode로도 디버그할 수 있음

“Rust Once, Run Everywhere”

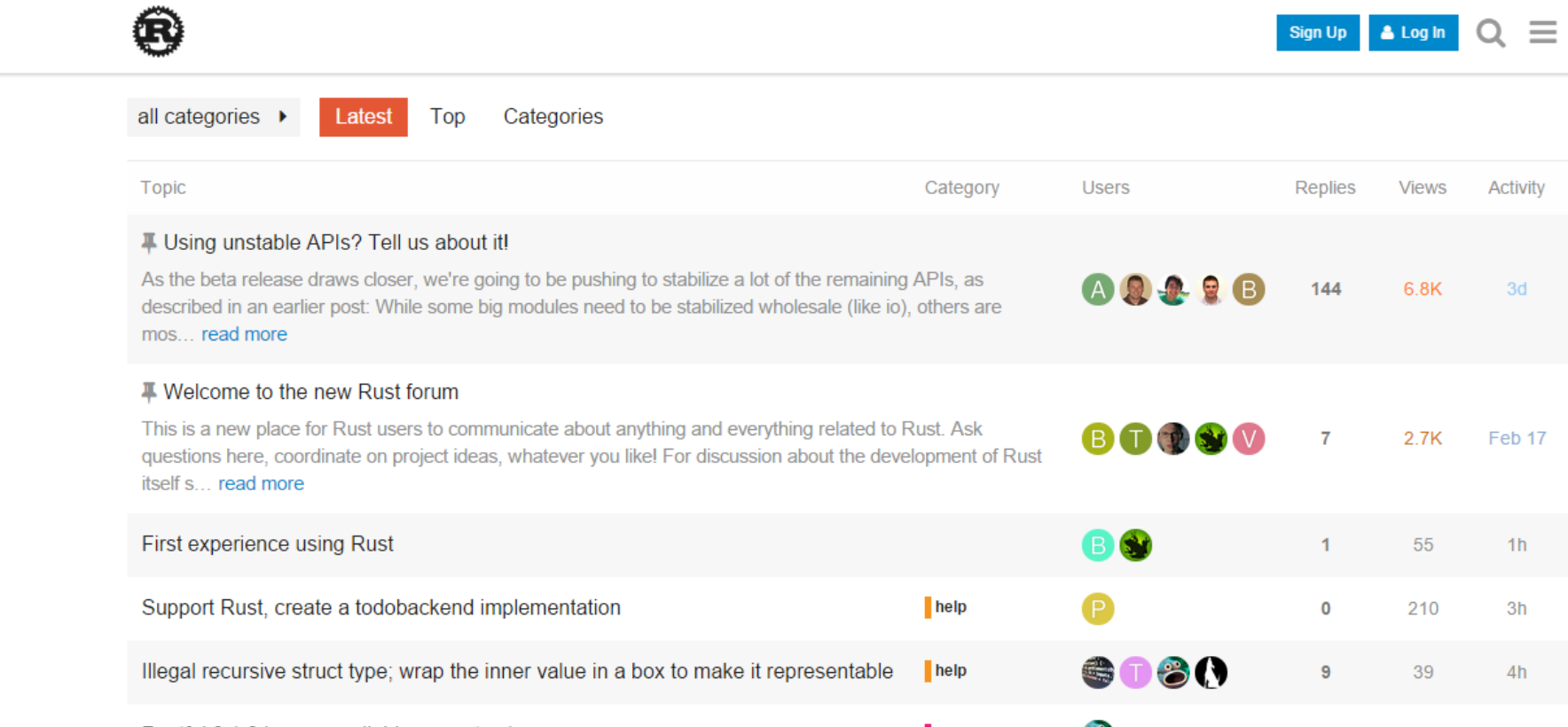
크로스 플랫폼


Fast | 빠르다
Safe | 안전하다
Easy | 쉽다


Fast | 빠르다
Safe | 안전하다
Easy | 쉽다
5월 15일 Rust 1.0.0 이 출시될 예정

좋은 커뮤니티

- 레딧, IRC, 자체 포럼
- #rust, #rust-gamedev, #rust-webdev, #servo, ...



rust-kr.org

- 한달에 한번씩 모여서 코딩모임을 함
- 학교 밖에 계시는 좋은 분들을 많이 만날수 있는 안 흔한 기회!

rust-kr.org - 한국 러스트 사용자 그룹

러스트(Rust)는 모질라(mozilla.org)에서 개발하고 있는, 메모리-안전하고 병렬 프로그래밍이 쉬운 차세대 프로그래밍 언어입니다. 아직 개발 단계이며 많은 기능이 구현 중으로, MIT/Apache2 라이선스로 배포됩니다.

한국 러스트 사용자 그룹은 러스트를 사용하거나 러스트 개발에 참여하고 있는 사용자 그룹입니다. 현재는 작은 규모이고 대부분 IRC를 이용하고 있습니다. 아래 대화창을 이용하여 지금 대화에 참가해 보세요! 궁금한 점을 물어보셔도 좋고 하고 싶은 이야기를 하셔도 좋습니다. 사생활 보호를 위해 웹 채팅에서는 단순한 대화 기능만 제공되므로, 채널에 직접 참가하시려면 IRC 클라이언트(hexchat, 윈도 / XChat Azure, 맥 / XChat, 리눅스)를 설치하고 irc://irc.ozinger.org/#rust로 접속하셔도 좋습니다. 찾아오신 것을 환영합니다!

- 설치하기

관련 링크 모음

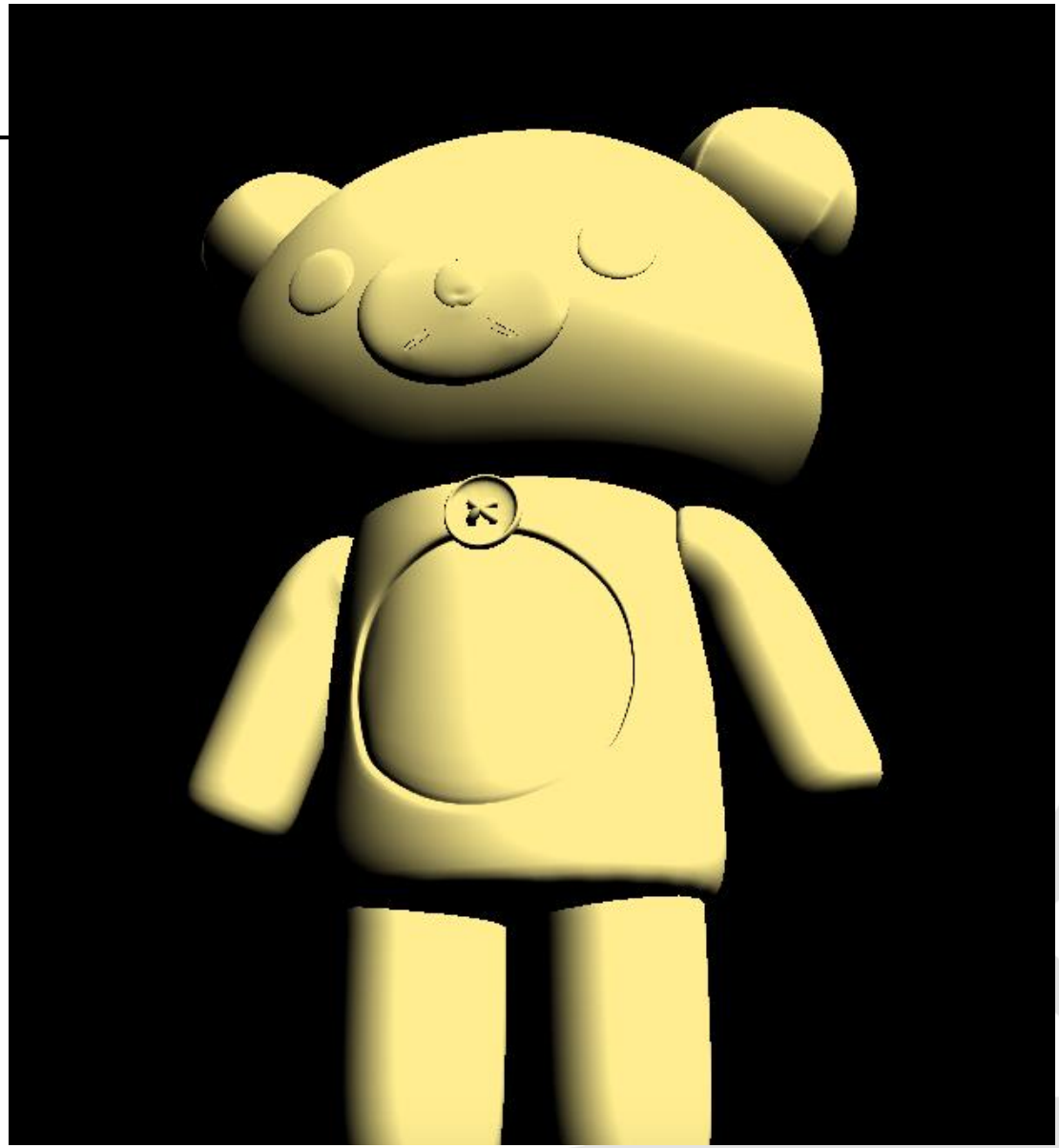
제안: UPnL 여름 Rust 스터디

- 먼저 Rust에 대해 배우고,
  각자, 혹은 같이 뭔가를 만들면서 진행하고자 함
- 기대효과
  - 시프에 익숙해지면 C/C++ 익히는데에도 훨씬 도움 많이 될거에염
  - 덜 잉여한 여름방학
  - 새내기의 경우 선배들을 뜯어먹을 수 있음
- 필참자
  - sgkim
  - apple
  - kinetic

내가 한것

- obj-rs

  Wavefront OBJ
  3D 모델 파서



근황

- Rust + OpenGL로 3D 대전액션게임 만드는중
- 빠름 + 크로스플랫폼
- sgkim이랑 같이하는중

- 관심있는 새내기는 연락하시오

Any Question?

김지현