main Go to file Code

mrconter1 add tuples and tuple destructur... f46ad5f · 3 days ago

src	add tuples and tuple d...	3 days ago
tests	add tuples and tuple d...	3 days ago
.gitignore	Add test suite with ru...	4 days ago
README.md	Update readme	3 days ago
rustc.php	add closure support vi...	4 days ago

README

rustc-php: A Rust compiler written in PHP

A Rust compiler written in PHP that emits x86-64 Linux ELF binaries directly (no LLVM, no assembler, no linker). Implements ownership checking, borrow checking, type checking, move semantics, generics, traits, closures, and iterators. Useful if you need to compile Rust on a shared hosting server from 2008 where the only installed runtime is PHP.

Installation

In order to execute Rust code you of course first need to install PHP. You can do this easily on Windows 11 by running:

```
winget install PHP.PHP.8.4
```

This compiler outputs valid machine code for Linux, so the most practical approach if you're on Windows is to use WSL. Start by installing Ubuntu (if you haven't already):

```
wsl --install
```

After the install completes, reboot your machine and then open Ubuntu from the Start menu to finish the initial setup.

Usage

Compile a `.rs` file by running:

```
php rustc.php main.rs -o main
```

Then execute the compiled binary through WSL:

```
wsl ./main
```

To see the exit code of the program:

```
wsl ./main; echo $?
```

What's supported

Types

- `i32`, `bool`, `u8`, `u16`, `u32`, `u64`, `u128`, `usize`
- `String` (heap string, move semantics)
- `&str` and string slice indexing (`s[0]`)
- `&T`, `&mut T` references with borrow checking
- Structs (named fields, field access, method calls)
- Enums with optional tuple payloads and `match`
- Unit type `()` in expressions, return types, and generic arguments
- Generics on functions, structs, and `impl` blocks
- Builtin `Option<T>` and `Result<T, E>` with `Some` / `None`, `Ok` / `Err` (no source definitions required)

Control flow

- `if` / `else` (including as expressions)
- `while`, `loop`, `break`, `continue`
- `for x in start..end` (range iteration)
- `match` with enum arms and wildcard `_`
- `return`

Functions and closures

- Free functions with multiple parameters and explicit return types
- `const fn` (accepted and treated as a regular function)
- `impl` blocks with `self`, `&self`, `&mut self`
- Trait definitions and `impl Trait for Type`
- Default trait method implementations
- Closures with capture-by-value (`|x: i32| x + captured_var`)

Ownership and borrowing

- Move semantics for non-`Copy` types
- Borrow and mutable borrow checking
- `Copy` inference for `i32`, `bool`, `&T`, and all-copy structs/enums

Modules and syntax

- `mod name;` declarations with file-based module resolution
- `pub` visibility on functions, structs, and fields
- `use` paths for cross-module imports
- Attributes `#[...]` and `#![...]` (parsed and skipped)

Output

- `println!("{}", expr)` — print a single value
- `exit(code)` — explicit exit with status code

Tests

Run the full test suite:

```
php tests/run.php
```

Test cases live in `tests/cases/` organized into `fundamentals/valid/`, `fundamentals/invalid/`, `modules/`, and `programs/`. Each `.rs` file declares its expected output in comments at the top:

```
// exit: 42
// stdout: hello
// error: Use of moved value
```

What's not yet implemented

Roughly in order of impact:

1. Compound assignment operators (`+=`, `-=`, `*=`, `/=`)
2. Tuples and tuple destructuring
3. `Vec<T>` and heap allocation
4. `f32` / `f64` floating point
5. `const` and `static` items
6. The `?` operator (Result is supported; `?` is not)

About

A Rust compiler with ownership checking, written in PHP

Readme
Activity
104 stars
0 watching
2 forks
Report repository

Contributors 1

mrconter1 Rasmus Lindahl

Languages

PHP 90.2% Rust 9.8%

7. Closures as function arguments (`fn apply(f: impl Fn(i32) -> i32)`)
8. Zero-parameter closures (`|| expr`)
9. Pattern matching beyond single-level enum variants
10. Multi-format `println!` (only `"{}"` with one argument is supported)
11. Lifetimes (borrow checker is simplified — no lifetime annotations)
12. Additional signed integer types (`i8` , `i16` , `i64` , `i128`)