Kirby Faverty Analysis Report

Version history:

v1: 2/2/25 Initial release with Report Milestone1 instructions

Purpose

The goal of the analysis report assignment is to give you experience
in systematically mastering the key features of a new programming
language.

Questions to Answer

- What is your chosen language?

The language that I have chosen is Rust.

- What range of applications/programming tasks is this language intended for?

Rust is primarely used as an alternative for a c or C++. It is a statically typed language
and a strongly typed language. This means that all of the varibles must be known at compilation
time. The Rust language is primarily used for developing high-preformance.
memory safe applications, ofter where speed and reliability are necessary.

- What (if any) distinctive features does the language have that
differentiate it from other languages? (What does the language documentation
brag about?)

Rust has been the most loved language in Stack Overflow for four years.
The reson for this is because of its distinct features one of those features
is known as a borrow checker. This is part of the compiler and is responsible for
making sure that the refrences do not outlive the data they refer to which helps
eliminate so many bugs caused by memory unsafety. Another distinct feature of Rust
is that it gives you a choice of storing the data. You can either store on the stack
or on the heap. Once that is determined Rust then determines at compile time when the
memory is no longer needed and can be cleaned up. This gives Rust the ability to be
way more effiecient.

- Who originally invented the language, and when?

Graydon Hoare founded Rust in 2006 as a personal project
while he was working at Mozilla.

- What are the main implementations of the language available today,
and who maintains them?

Rustc = The official Rust compiler and maintained by the Rust team and
funded and backed by the Rust Foundation.

MRustC = An alternative to the Rust compiler foocused on bootstrapping Rust
without requiring Rust itself. This implementation is maintained by a single
developer themskr and is used for academic and niche purposes

Rust-gcc = An experimental Rust front end for GCC which is used to integrate
Rust in the GNU Compiler Collection. This is being developed by the GCC Rust
Project, with contributions from Rust and GCC communities.

- Which exact implementation/version of the language are you reporting
on here?

I will be reporting on the Rustc which is the official Rust compiler. The
version of Rustc that I will be reporting on is the current version, which
is 1.84.1.

- What authoritative documentation (reference manuals, etc.) is available
for the language/implementation?

The main documantation that I use for Rust is stack overflow, Rust by example, and geeks for
geeks.
However there are so many resources all over the internet and in books as Rust is
a very popular language.

- Is the language officially standardized by some industry group or
government organization? If so, include a pointer to the relevant standards
document.

Rust is not oficially standardized by a certain industry or groups. However it
goverened by the Rust Foundation. Which oversees its development and updates.

Here is a link to the Rust foundation:

https://Rustfoundation.org/

- Is the language typically compiled or interpreted? What about the
implementation you are reporting on here? If the implementation includes

multiple stages (e.g. compilation to an intermediate code that is interpreted),
describe this.

Rust is a compiled language. Rustc (the compiler that I am addressing)
is compiled as follows:

1. Parsing and AST Generation -
   Just like Python3 and gcc Rustc parses the
   source code and created an Abstract Syntax Tree

2. Hir, MIR, LLVM - The AST is then tranformed into
HIR (Highy Level Intermediate Representation) which is the program
code is represented in a form that is closer to the original source code.
It is then Mid Level Intermediate Representation, which optimizes the structure.
It is then furthered lowered to LLVM Intermediate Representation which generates
and optimizes the machine code.

3. Linking - The machine code that was created from the LLVm is executed in Binary.

- Is there an official formal grammar (e.g. in BNF) for the language?
If so, include a pointewr to this if possible.

Primitive types and expressions

- Does the language use static or dynamic typing?

Rust is a statically typed language.

- What kinds of primitive numeric types are supported (integers, floats, etc.)? What are their ranges and precisions?`

The numeric types in Rust are integers which are signed and unsigned and support
up to 64-bits.

To declare an int it is a little different from C++ or python. Instead of just say int you must either do i32 a signed 32 bit integer. Or a u32 an unsigned 23 bit integer. The same goes for float. Floating Point which are f23 and f64 repectively.

Numeric Operations which are the basic mathematical operation you would
see for all number types (such as +, -, *, /)

boolean types which are two possible values true, or false

Rust supports a wide vareity of character types such as char, and string
and also supports arrays and tuples and so forth.

- What happens if integer operations overflow? What happens on division
by zero?

When an integer overflow occurs Rust panics and crashed. This will
also happens to division of zero as well. This occurs because Rust focuses on safety
so rather than giving an undifined value the program the program will just crash, and
print out an error.

- How are booleans represented? Are booleans distinct from integers?
What values are considered "truthy" and "falsy" in conditional tests?

There are two boolean expressions just like I stated above. They are both
true, and false. Unlike C++ they cannot be ascciociated to a value unless
excicitly converted.

- How are strings represented? How are basic string operations (substring,
concatenation, etc.) performed?

Strings are represented as &str and example of that is below:

```rust
fn main(){
   let name:&str = "Kirby";
   println!("hello you rname is {}", name)
}
```

As you can see &str is a type string. Rust also support basic string operations
and they are very similar to python and the string library in c++. For example
concatenation uses the + or the push_str method. One intresting feature of string
in Rust is that strings can either be immutable or mutable depending on how you declare
them.

```rust
(an example of concatenation)
fn main(){
   let mut s = String::from("Hello");
   s.push_str(", Kirby");
   println!("{}", s);
}
```

output: Hello, Kirby

All of the other string basic strings work the same way. One important factor to know is that you need to make sure if you wan to edit a string that it needs to be mutable. Otherwise if it is immutable an error will pop up.

- What operators are allowed in expressions? What are any interesting features of the precedence and associativity rules for operators (e.g., differences from other languages like C++ or Python)?

The operators that are allowed in expressions are very similar to c++ and python those are arithmatic operations, comparision Operators, logical operators, bitwise operators, and assignment operators.

The difference between Rust and other languages (specifically c++ and python) is that ++ and -- so increment and decrement are not supported in Rust. However .. is used as a range operator, and ! are used for calling macros, like in the above example println!. another difference that I found was that there is no terenary conditional operator (condition ? expr1 : expr2) instead it uses if expressions

Rust : letx = if condition {"true" } else {"false"}

The last difference that I found was that bitwise operators (e.x. &, <<, >>) in Rust work more with integers providing better safety when using theese operations.

- Is there an equivalent to `let` bindings in expressions?

The equivalent to the 'let' binding in rus is the same you can use let freely in expressions. Here is an example of let bindings being used in a simple funciton.

```rust
fn main(){
    let mut sum = 0;
    for i in 1..5{
        let num = i * i;
        sum += num;
    }
    println!("This is the resut: {}", sum)
}
```

Functions, binding, scoping

- What is the syntax for function definitions? What about for mutually recursive function definitions?

Funciton definition starts as follows:

```rust
fn multiple(a:i32,b:i32) -> i32
{
    return a * b;
}
```

This is the basic funciton syntax for multiplying to numbers and returning them. For mutable recursive function it is very similar to C++. Below is an example of a recursive function in Rust that finds the factorial of a number.

```rust
fn fact(num: i32) -> i32{
    if num == 0{
        return 1;
    }
    else{
        return num * fact(num-1);
    }
}

fn main(){
    let num: i32 = 5;
    let sum: i32 = fact(num);
    println!("This is the resut: {}", sum);
}
```

- Can function definitions be nested (i.e. can we define a local function within the body of another function)? If so, what is the syntax for this?

In Rust unfortunately you cannot do nested functions(i.e functions inside of other functions). However you can achieve similar results nested function definitions with closures(which is Rusts version of a a lamda function). Below is an example of a closure.

```rust
```

```
fn main(){
  let closure = |x:i32, y:i32| x + y;
  let result:i32 = closure(5,4);

   println!("This is the result: {}", result);
}
```

The closure acts a little like a nested function however, with different synax and no return statement.

- Are there any unusual features to the binding and scoping rules (e.g. like Python's `global` and `nonlocal` declarations)?

The most unusual thing about global declerations in Rust is that unlike python Rusts Global varies myst be static. Rust also has some unique bindings such as varible shadowing this means that you can redeclare a varible in the same scope. Another weird thing that Rust does Ownership and Borrowing. Rust controls memory safely without the need of a garbage collector.

- Can functions be passed as arguments to other functions (e.g. passing a comparison operator to a general-purpose `sort` function)?

Yes, Rust allows the passing of function pointer to function an example of this is

```rust
fn add_one(x: i32) -> i32 {
   x + 1
}

fn apply_function(f: fn(i32) -> i32, value: i32) -> i32 {
   f(value)  // Call the passed function
}

fn main() {
   let result = apply_function(add_one, 2);
   println!("Result: {}", result);

//Output: 3
}
```

~~~
```
~~~

As you can see in the function apply_function calls fn(i32) -> i32 which is the function above that. This function adds one to 2 and outputs 3.

- Can functions be stored in data structures (e.g. when serving as a call-back)?

Yes functions can be stored in data structure. Below is a provided example of this.

```rust
struct Button {
    on_click: Box<dyn Fn()>,
}

impl Button {
    fn click(&self) {
        (self.on_click)();
    }
}

fn main() {
    let button = Button {
        on_click: Box::new(|| println!("Button clicked!")),
    };

    button.click();  // Output: Button clicked!
    button.click();
}

```

As you can see this function is created for a button click.
Whenever a button.click is called. However it is a little
different than c++ or python because the function is stored
on the impl which is short for implementation.This is used to
define methodsfor structs, traits, and types. In this case
and in many casses it is used to add methods to a struct.

- Is there support for anonymous functions (lambda expressions)? If so, what is the syntax for them? Are there restrictions on the form of anonymous function bodies?

Lambda functions are called closures in Rust. I explained closure functions above but the principle is if you wanted to make lambda functions in Rust the syntax is like this:

```rust
fn main(){
  let closure = |x:&str, y:&str| x.to_owned() + y;
  let result = closure("Hello", ", World");

   println!("This is the result: {}", result);
}
```

This is an example of a closure to concatenate two strings. This is the equivalent of a lambda function in Python.

- Does the language use static or dynamic binding (in the sense discussed in class)?

Rust uses Static binding and is determined in compile time. This means that the exact method to call is known when the code is compiled. However the cool part about Rust is that it can also supports dynamic binding. The method of dynamic binding is that the method call is resolved at runtime. It is more flexible allowing for more generic and polymorphic code.

- How are arguments passed to functions (e.g., by value, by name, something else)?

There are three ways that arguments can be passed to functions.

1. Pass by value: This means that when passed the function takes ownership of that varible and cannot be changed.

2. Pass by Refrence: This means that the ownership of the value is borrowed and you are not giving the ownership to the called and in the end the caller retains control of the varible

3. The final one is passing by mutable reference: This means that the varible can be modified in the calling scope. You cannot have more than one mutable refrence oof the same data at the same time.

##
#########################

Additional Questions to Answer for Final Report

For scoring, all questions are weighted equally, except for those marked "[2x points]" which are weighted twice as much as the rest.

Statements and Control

- What primitive or atomic statements (i.e. statements that do not have sub-statements) does the language provide? Can expressions be used as atomic statements?

- Can assignment be performed as a side-effect of evaluating an expression, or only by a statement? (For example, in C++, `(x = 1) + 2` assigns to `x` in the middle of an expression.)

- What structured control statements does the language provide for sequencing? (For example, blocks and `;` in C++.)

- What structured control statements does the language provide for selection? (For example, `if` and `switch` in C++.)

- What structured control statements does the language provide for iteration? (For example, `for` and `while` in C++.)

- Does the language support a notion of iterators? If so, can programmers define their own iterators?

- Does the language support general `goto` statements or some restricted form of them (such as labeled `break`s)?

- Does the language support exceptions? If so, how are they defined, thrown, and caught?

Data structures

- [2x points] What kinds of simple product types (e.g. records, tuples, classes) are supported? Are they mutable or immutable (or can individual fields be declared mutable or immutable)? Are they boxed or unboxed? Homogeneous or heterogeneous (i.e., must all fields be the same type, or can they be different types)? Can indices for accessing elements be computed at runtime, or must they be static?

- [2x points] What kinds of array and/or dictionary types are supported? Are they homogeneous or heterogeneous (i.e., must all indices/values be the same type, or can they be different types)? What types of values can be used as indices? Can these structures be resized dynamically, or is their size fixed at creation time?

- Can the programmer define sum types (e.g., types composed of multiple alternatives), and in particular an option type? If the language doesn't provide explicit support for them, how might they simulated?

- What would be the most idiomatic way to define an AST tree type (like the one we have used in our interpreters) in this language?

- For products and other data structures, does the language support reference equality or structural equality, or both?

- Does the language support garbage collection? If so, what type of collector is used (by the implementation you're studying)? If not, how is heap memory deallocated?

- Is it possible to have call-by-reference function parameters? If so, how are these specified?

- Can function values be returned by (other) functions? If so, are there any restrictions on this feature?

Type systems

- Does the language support polymorphic or generic types for functions and data structures? If so, how are these specified and used?

- Is there any type inference (e.g., `auto` in C++)? If so, in what contexts (function definitions, local variables, function applications, etc.) does it work?

- Does the language use other kinds of static analysis beyond type checking (e.g. checking that all paths return a value, or that all variables are initialized before use)?

Objects and Modules

- [2x points] Does the language support Object-Oriented Programming (OOP) explicitly, or provide a way to simulate it? If so, how are objects defined? Is there a way to encapsulate object data? Is there support for dynamic method dispatch? Is there support for inheritance and sub-typing?

- [2x points] Does the language have some form of modules? If so, how are they defined? How does the programmer distinguish public vs. private parts of a module? How are module definitions imported into other modules?