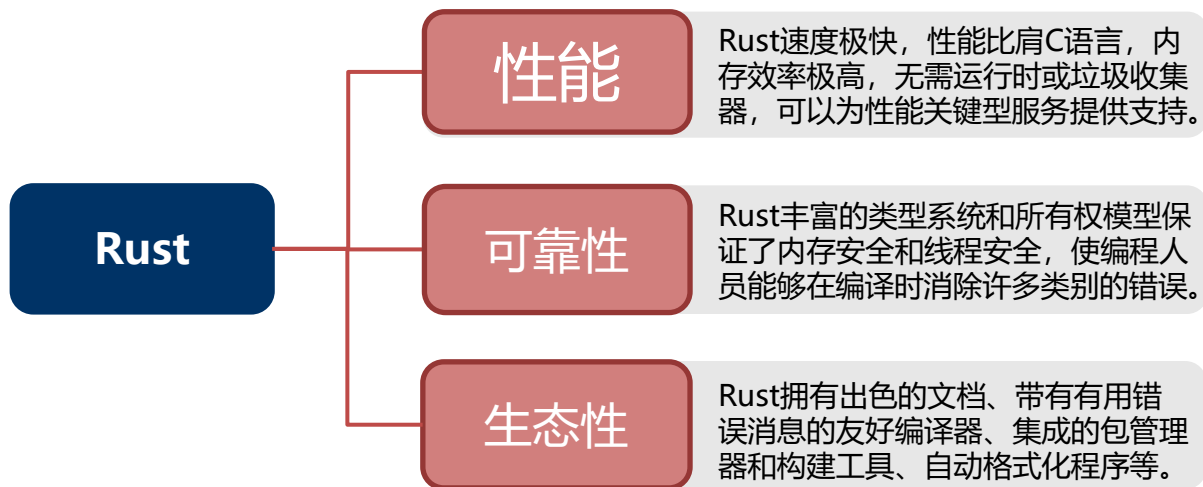


类Rust编译器实现



Rust是一门能够安全高效地编写系统级程序的语言，在内存安全方面具有显著优势。



知名问答平台StackOverflow的调查表明，自2015年以来，Rust一直是开发者最爱的编程语言。

nature

《Nature》杂志2020年尾的文章《Why Scientists are Turning to Rust》中也强调：科学家极为推崇Rust。

华为是五位Rust基金会创始白金会员成员之一，引领了Rust语言的发展。

华为内部使用

- Rust 广泛用于嵌入式系统开发、系统驱动、云计算、虚拟存储、网络传输协议、并发编程框架基础库等产品中。
- 华为实验室正基于 Rust 探索先进的代码解析、安全分析等工具。

Rust社区贡献

- 华为深度参与了在Rust社区中，贡献了包括C到Rust转换、内联汇编、交叉编译、Parking Lot并发库、SIMD 基础库、文档导航、代码多态化、热补丁、AOP等特性。

Rust中国推广

- 华为战略支持了在中国举办的Rust China Conf大会，并推行多项社区活动。
- 华为也为中国的开发者提供Rust教程和Rust编码规范等。

大作业1：词法和语法分析工具设计与实现

- 了解类Rust词法、语法规则
- 基本功能：对类Rust示例程序实现词法和语法分析，并输出分析结果
- 扩展功能不限
- 分组，每组2至3人
- 撰写设计和说明文档

类Rust词法规则

- 关键字: `i32` | `let` | `if` | `else` | `while` | `return` | `mut` | `fn` | `for` | `in` | `loop` | `break` | `continue`
- 标识符: `(字母|_)(字母|数字|_)*` (注: 不与关键字相同)
- 数值: `数字(数字)*`
- 赋值号: `=`
- 算符: `+` | `-` | `*` | `/` | `==` | `>` | `>=` | `<` | `<=` | `!=`
- 界符: `(` | `)` | `{` | `}` | `[` | `]`
- 分隔符: `;` | `:` | `,`
- 特殊符号: `->` | `.` | `..`
- 注释号: `/*` | `*/` | `//`
- 字母: `|a|...|z|A|...|Z|`
- 数字: `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` |
- 结束符: `#`

类Rust语法规则

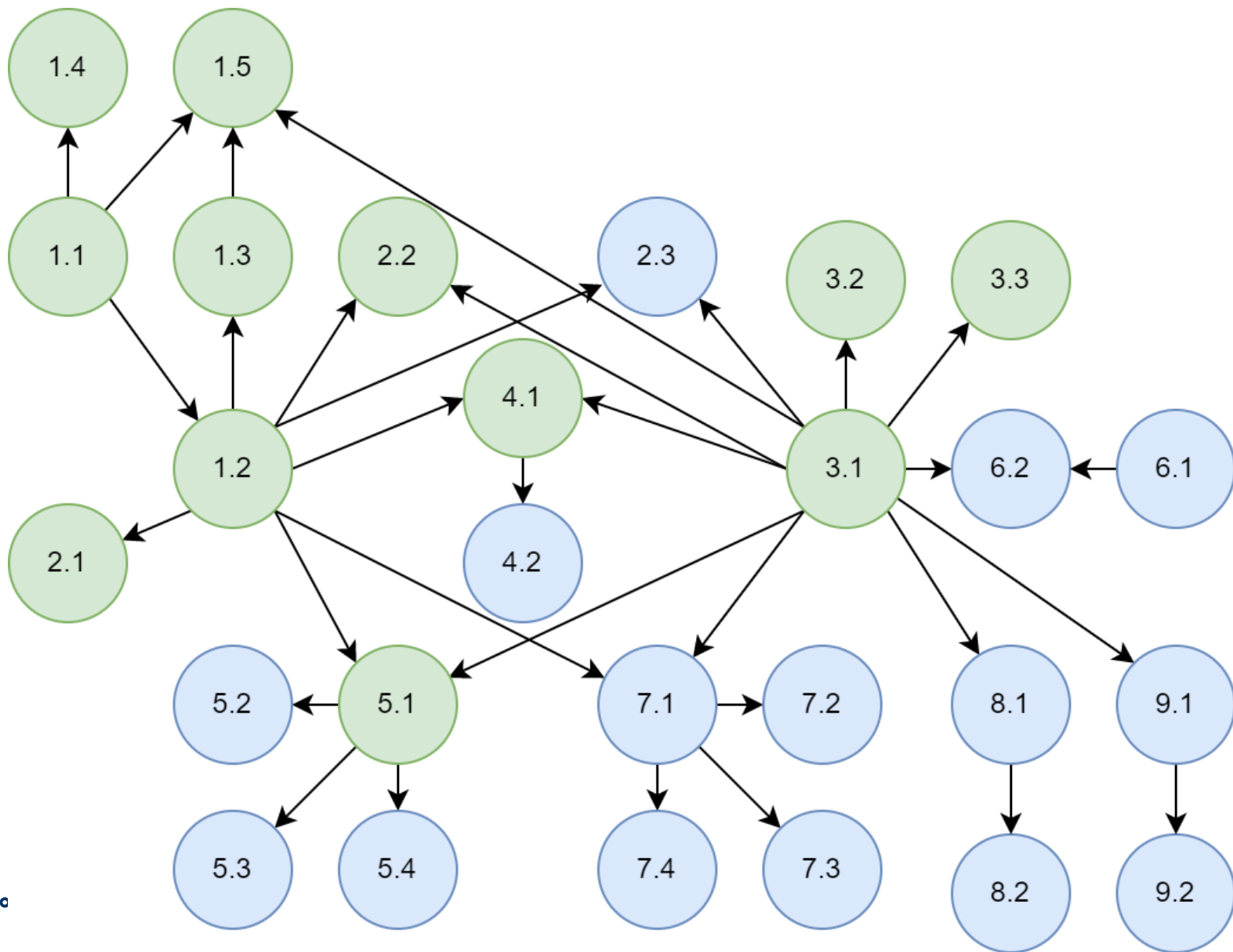
绿色节点为基础规则：

0.1、0.2、0.3、
1.1、1.2、1.3、1.4、1.5、
2.1、2.2、2.3、
3.1、3.2、3.3、
4.1、5.1。

蓝色节点为拓展规则：

4.2、5.2、5.3、5.4、
6.1、6.2、
7.1、7.2、7.3、7.4
8.1、8.2
9.1、9.2。

注：0.1、0.2、0.3未在图中画出。



类Rust语法规则

➤ 1.1 基础程序

- Program -> <声明串>
- <声明串> -> 空 | <声明> <声明串>
- <声明> -> <函数声明>
- <函数声明> -> <函数头声明> <语句块>
- <函数头声明> -> fn <ID> '(' <形参列表> ')'
- <形参列表> -> 空
- <语句块> -> '{' <语句串> '}'
- <语句串> -> 空

```
fn program_1_1() {  
  
}
```

➤ 0.1 变量声明内部

- <变量声明内部> -> mut <ID>

➤ 0.2 类型

- <类型> -> i32

➤ 0.3 可赋值元素

- <可赋值元素> -> <ID>

类Rust语法规则

➤ 1.2 语句（前置规则1.1）

- $\langle \text{语句串} \rangle \rightarrow \langle \text{语句} \rangle \langle \text{语句串} \rangle$
- $\langle \text{语句} \rangle \rightarrow \text{';'}$

➤ 1.3 返回语句（前置规则1.2）

- $\langle \text{语句} \rangle \rightarrow \langle \text{返回语句} \rangle$
- $\langle \text{返回语句} \rangle \rightarrow \text{return '}'$

➤ 1.4 函数输入（前置规则0.1、0.2、1.1）

- $\langle \text{形参列表} \rangle \rightarrow \langle \text{形参} \rangle \mid \langle \text{形参} \rangle \text{' ' } \langle \text{形参列表} \rangle$
- $\langle \text{形参} \rangle \rightarrow \langle \text{变量声明内部} \rangle \text{' :' } \langle \text{类型} \rangle$

➤ 1.5 函数输出（前置规则0.2、1.1、1.3、3.1）

- $\langle \text{函数头声明} \rangle \rightarrow \text{fn } \langle \text{ID} \rangle \text{' (' } \langle \text{形参列表} \rangle \text{')' ' } \rightarrow \text{' } \langle \text{类型} \rangle$
- $\langle \text{返回语句} \rangle \rightarrow \text{return } \langle \text{表达式} \rangle \text{' ;'}$

```
fn program_1_2() {  
    .....  
}
```

```
fn program_1_3() {  
    return ;  
}
```

```
fn program_1_4(mut a:i32) {  
  
}
```

```
fn program_1_5() -> i32 {  
    return 1;  
}
```

类Rust语法规则

➤ 2.1 变量声明语句（前置规则0.1、0.2、1.2）

- $\langle \text{语句} \rangle \rightarrow \langle \text{变量声明语句} \rangle$
- $\langle \text{变量声明语句} \rangle \rightarrow \text{let } \langle \text{变量声明内部} \rangle \text{ ':' } \langle \text{类型} \rangle \text{ ';' }$
- $\langle \text{变量声明语句} \rangle \rightarrow \text{let } \langle \text{变量声明内部} \rangle \text{ ';' }$

```
fn program_2_1() {  
    let mut a:i32;  
    let mut b;  
}
```

➤ 2.2 赋值语句（前置规则0.3、1.2、3.1）

- $\langle \text{语句} \rangle \rightarrow \langle \text{赋值语句} \rangle$
- $\langle \text{赋值语句} \rangle \rightarrow \langle \text{可赋值元素} \rangle \text{ '=' } \langle \text{表达式} \rangle \text{ ';' }$

```
fn program_2_2(mut a:i32) {  
    a=32;  
}
```

➤ 2.3 变量声明赋值语句（前置规则0.1、0.2、1.2、3.1）

- $\langle \text{语句} \rangle \rightarrow \langle \text{变量声明赋值语句} \rangle$
- $\langle \text{变量声明赋值语句} \rangle \rightarrow \text{let } \langle \text{变量声明内部} \rangle \text{ ':' } \langle \text{类型} \rangle \text{ '=' } \langle \text{表达式} \rangle \text{ ';' }$
- $\langle \text{变量声明赋值语句} \rangle \rightarrow \text{let } \langle \text{变量声明内部} \rangle \text{ '=' } \langle \text{表达式} \rangle \text{ ';' }$

```
fn program_2_3() {  
    let mut a:i32=1;  
    let mut b=1;  
}
```

类Rust语法规则

➤ 3.1 基本表达式（前置规则0.3）

- $\langle \text{语句} \rangle \rightarrow \langle \text{表达式} \rangle ;$
- $\langle \text{表达式} \rangle \rightarrow \langle \text{加法表达式} \rangle$
- $\langle \text{加法表达式} \rangle \rightarrow \langle \text{项} \rangle$
- $\langle \text{项} \rangle \rightarrow \langle \text{因子} \rangle$
- $\langle \text{因子} \rangle \rightarrow \langle \text{因子} \rangle$
- $\langle \text{元素} \rangle \rightarrow \langle \text{NUM} \rangle \mid \langle \text{可赋值元素} \rangle \mid '(' \langle \text{表达式} \rangle ')'$

```
fn program_3_1__1() {  
    0;  
    (1);  
    ((2));  
    (((3)));  
}
```

```
fn program_3_1__2(mut a:i32) {  
    a;  
    (a);  
    ((a));  
    (((a)));  
}
```


类Rust语法规则

➤ 3.2 表达式增加计算和比较（前置规则3.1）

- $\langle \text{表达式} \rangle \rightarrow \langle \text{表达式} \rangle \langle \text{比较运算符} \rangle \langle \text{加法表达式} \rangle$
- $\langle \text{加法表达式} \rangle \rightarrow \langle \text{加法表达式} \rangle \langle \text{加减运算符} \rangle \langle \text{项} \rangle$
- $\langle \text{项} \rangle \rightarrow \langle \text{项} \rangle \langle \text{乘除运算符} \rangle \langle \text{因子} \rangle$
- $\langle \text{比较运算符} \rangle \rightarrow '<' \mid '<=' \mid '>' \mid '>=' \mid '==' \mid '!='$
- $\langle \text{加减运算符} \rangle \rightarrow '+' \mid '-'$
- $\langle \text{乘除运算符} \rangle \rightarrow '*' \mid '/'$

```
fn program_3_2() {  
    1*2/3;  
    4+5/6;  
    7<8;  
    1*2+3*4<4/2-3/1;  
}
```

➤ 3.3 函数调用（前置规则3.1）

- $\langle \text{元素} \rangle \rightarrow \langle \text{ID} \rangle '(' \langle \text{实参列表} \rangle ')'$
- $\langle \text{实参列表} \rangle \rightarrow \text{空} \mid \langle \text{表达式} \rangle \mid \langle \text{表达式} \rangle ',' \langle \text{实参列表} \rangle$

```
fn program_3_3_1() {  
  
}  
fn program_3_3_2() {  
    program_3_3_1();  
}
```

类Rust语法规则

➤ 4.1 选择结构（前置规则1.2、3.1）

- $\langle \text{语句} \rangle \rightarrow \langle \text{if语句} \rangle$
- $\langle \text{if语句} \rangle \rightarrow \text{if } \langle \text{表达式} \rangle \langle \text{语句块} \rangle \langle \text{else部分} \rangle$
- $\langle \text{else部分} \rangle \rightarrow \text{空} \mid \text{else } \langle \text{语句块} \rangle$

➤ 4.2 增加else if（前置规则4.1）

- $\langle \text{else部分} \rangle \rightarrow \text{else if } \langle \text{表达式} \rangle \langle \text{语句块} \rangle \langle \text{else部分} \rangle$

```
fn program_4_1(a:i32) -> i32 {  
    if a>0 {  
        return 1;  
    } else {  
        return 0;  
    }  
}
```

```
fn program_4_2(a:i32) -> i32 {  
    if a>0 {  
        return a+1;  
    } else if a<0 {  
        return a-1;  
    } else {  
        return 0;  
    }  
}
```

类Rust语法规则

➤ 5.1 while循环结构（前置规则1.2、3.1）

- <语句> -> <循环语句>
- <循环语句> -> <while语句>
- <while语句> -> while <表达式> <语句块>

```
fn program_5_1(mut n:i32) {  
    while n>0 {  
        n=n-1;  
    }  
}
```

➤ 5.2 for循环结构（前置规则5.1）

- <循环语句> -> <for语句>
- <for语句>-> for <变量声明内部> in <可迭代结构> <语句块>
- <可迭代结构> -> <表达式> '..' <表达式>

```
fn program_5_2(mut n:i32) {  
    for mut i in 1..n+1 {  
        n=n-1;  
    }  
}
```

➤ 5.3 loop循环结构（前置规则5.1）

- <循环语句> -> <loop语句>
- <loop语句>->loop <语句块>

```
fn program_5_3() {  
    loop {  
  
    }  
}
```

类Rust语法规则

➤ 5.1 while循环结构（前置规则1.2、3.1）

- `<语句> -> <循环语句>`
- `<循环语句> -> <while语句>`
- `<while语句> -> while <表达式> <语句块>`

```
fn program_5_4_1() {  
    while 1==0 {  
        continue;  
    }  
}
```

➤ 5.4 增加break和continue（前置规则5.1）

- `<语句> -> break ';' | continue ';' |`

```
fn program_5_4_2() {  
    while 1==1 {  
        break;  
    }  
}
```

类Rust语法规则

➤ 6.1 声明不可变变量（前置规则0.2）

- `<变量声明内部> -> <ID>`

➤ 6.2 借用和引用（前置规则3.1、6.1）

- `<因子> -> '*' <因子> | '&' mut <因子> | '&' <因子>`
- `<类型> -> '&' mut <类型> | '&' <类型>`

`&mut` : 可变引用

`&` : 不可变引用

`*` : 解引用

```
fn program_6_2_1() {  
    let mut a:i32=1;  
    let mut b:&mut i32=&mut a;  
    let mut c:i32=*b;  
    *b=2;  
}
```

```
fn program_6_2_2() {  
    let a:i32=1;  
    let b:& i32=&a;  
    let c:i32=*b;  
}
```

```
fn program_6_1() {  
    let a:i32;  
    let b;  
    let c:i32=1;  
    let d=2;  
}
```

类Rust语法规则

➤ 7.1 函数表达式块（前置规则1.2、3.1）

- $\langle \text{表达式} \rangle \rightarrow \langle \text{函数表达式语句块} \rangle$
- $\langle \text{函数表达式语句块} \rangle \rightarrow \{ \langle \text{函数表达式语句串} \rangle \}$
- $\langle \text{函数表达式语句串} \rangle \rightarrow \langle \text{表达式} \rangle \mid \langle \text{语句} \rangle \langle \text{函数表达式语句串} \rangle$

➤ 7.2 函数表达式块作为函数体（前置规则7.1）

- $\langle \text{函数声明} \rangle \rightarrow \langle \text{函数头声明} \rangle \langle \text{函数表达式语句块} \rangle$

```
fn program_7_1(mut x:i32,mut y:i32) {  
    let mut z={  
        let mut t=x*x+x;  
        t=t+x*y;  
        t  
    };  
}
```

```
fn program_7_2(mut x:i32,mut y:i32) -> i32 {  
    let mut t=x*x+x;  
    t=t+x*y;  
    t  
}
```

类Rust语法规则

➤ 7.3 选择表达式（前置规则7.1）

- `<表达式> -> <选择表达式>`
- `<选择表达式> -> if <表达式> <函数表达式语句块> else <函数表达式语句块>`

➤ 7.4 循环表达式（前置规则7.1）

- `<表达式> -> <loop语句>`
- `<语句> -> break <表达式> ';' ;`

```
fn program_7_3(mut a:i32) {  
    let mut b=if a>0 {  
        1  
    } else {  
        0  
    };  
}
```

```
fn program_7_4() {  
    let mut a=loop {  
        break 2;  
    };  
}
```

类Rust语法规则

➤ 8.1 数组（前置规则0.2、3.1）

- $\langle \text{类型} \rangle \rightarrow '[' \langle \text{类型} \rangle ';' \langle \text{NUM} \rangle ']$
- $\langle \text{因子} \rangle \rightarrow '[' \langle \text{数组元素列表} \rangle ']' \mid \langle \text{数组元素} \rangle$
- $\langle \text{数组元素列表} \rangle \rightarrow \text{空} \mid \langle \text{表达式} \rangle \mid \langle \text{表达式} \rangle ',' \langle \text{数组元素列表} \rangle$

```
fn program_8_1() {  
    let mut a:[i32;3];  
    a=[1,2,3];  
}
```

➤ 8.2 数组元素（前置规则8.1）

- $\langle \text{可赋值元素} \rangle \rightarrow \langle \text{元素} \rangle '[' \langle \text{表达式} \rangle ']$
- $\langle \text{可迭代结构} \rangle \rightarrow \langle \text{元素} \rangle$

```
fn program_8_2(mut a:[i32;3]) {  
    let mut b:i32=a[0];  
    a[0]=1;  
}
```


类Rust语法规则

➤ 9.1 元组（前置规则0.2、3.1）

- $\langle \text{类型} \rangle \rightarrow '(' \langle \text{元组类型内部} \rangle ')'$
- $\langle \text{元组类型内部} \rangle \rightarrow \text{空} \mid \langle \text{类型} \rangle ',' \langle \text{类型列表} \rangle$
- $\langle \text{类型列表} \rangle \rightarrow \text{空} \mid \langle \text{类型} \rangle \mid \langle \text{类型} \rangle ',' \langle \text{类型列表} \rangle$
- $\langle \text{因子} \rangle \rightarrow '(' \langle \text{元组赋值内部} \rangle ')'$
- $\langle \text{元组赋值内部} \rangle \rightarrow \text{空} \mid \langle \text{表达式} \rangle ',' \langle \text{元组元素列表} \rangle$
- $\langle \text{元组元素列表} \rangle \rightarrow \text{空} \mid \langle \text{表达式} \rangle \mid \langle \text{表达式} \rangle ',' \langle \text{元组元素列表} \rangle$

```
fn program_9_1() {  
    let a:(i32,i32,i32);  
    a=(1,2,3);  
}
```

➤ 9.2 元组元素（前置规则8.1）

- $\langle \text{可赋值元素} \rangle \rightarrow \langle \text{因子} \rangle '.' \langle \text{NUM} \rangle$

```
fn program_9_2(mut a:(i32,i32)) {  
    let mut b:i32=a.0;  
    a.0=1;  
}
```

■ if123

- 应该能够识别为标识符

■ if=123

- 应该识别为三个单词：
 - (if, 保留字)
 - (=, 算符)
 - (123, 整数)

评分标准

实验评价内容	所占比重	要求
问题分析能力	20%	说明词法分析或语法分析原理，绘制必要的状态转换图。
系统方案（算法）设计能力	20%	报告中体现系统各模块的总体设计和详细设计。
编程能力	20%	独立编程实现要求的全部功能，正确无误。
撰写报告能力	30%	表达通顺、结构清晰、内容完整、实验充分、提出个人想法，不存在抄袭。
查阅文献资料能力	10%	报告中列出所查阅的文献资料，含图书、论文、网络资源等。

报告要求

- **设计文档1份**
- **程序源代码、可执行代码1份**
- **程序实例与结果截屏**
- **报告PPT1份**

写在最后

➤ 部分羁绊

- 2.2 赋值语句 + 拓展<类型>的规则（如8.1）：类型多于一种，变量赋值时需要检测类型是否对应，不对应应该报错。
- 5.2 for循环结构 + 8.1 数组：数组是可迭代结构，可以被for循环访问；而其他类型的变量不可迭代，被for循环访问时应报错。
- 2.2 赋值语句 + 6.1 声明不可变变量 + 循环的规则（如5.1）：若循环中中存在对不可变变量进行赋值的操作，应该报错。

➤ 说明

- 可以也鼓励大家对本文档中的产生式进行改写，也可以拓展更多未在文档中提到的Rust的语法规则。
- 本规则中存在极个别的羁绊会使得LR(1)无法处理，如果发现并对此进行思考（如能否改写产生式来解决问题，如果不能则分析原因），将作为加分项。

➤ 思考

- 1.4 函数输入中的形参列表可以识别怎样的语言？
- 9.1 元组 比 8.1 数组 多了几条产生式，思考原因？