

Name: 王何佳

Student ID: 2023211603

Class Number: 2023211804

Container ID: af334d0e5bb7b543090c0922e7af44654b94be08a5134f7ece8c27ec1384dd11

Lab 4

一、实验目的

1. **深入理解错误处理：**掌握编译器前端在语法分析阶段进行错误处理的基本机制和重要性。
2. **掌握 Bison 的错误恢复：**学习使用 Bison/Yacc 中的 `error` 关键字来捕获语法错误，并定义恢复规则，以增强编译器的鲁棒性。
3. **实践 JSON 语法分析：**针对 JSON 这一具体的数据格式，编写语法规则来识别并报告各种常见的格式错误，如括号不匹配、缺少逗号、多余的值等。
4. **解决语法冲突：**学习识别和解决在添加错误恢复规则时可能引入的移入/规约冲突(shift/reduce conflicts)。

二、实验原理

本次实验的核心是语法分析阶段的**错误恢复(Error Recovery)**。当语法分析器(Bison)遇到一个不符合任何语法规则的 Token 时，它会进入“错误模式”。

1. **error 标记：**Bison 允许在产生式中插入特殊的 `error` 标记。当分析器处于错误模式时，它会丢弃堆栈上的状态和输入 Token，直到找到一个允许 `error` 标记的状态和 Token。此时，分析器会移入 `error` 标记，并执行相应的动作，然后尝试从 `error` 标记之后的状态继续正常解析。
2. **自定义错误信息：**默认情况下，Bison 在出错时会调用 `yyerror` 函数。在本次实验中，为了提供更精确的错误定位，我们通过在 `error` 规则中直接 `printf` 等方式输出自定义信息，并有意地使 `yyerror` 函数保持“沉默”，避免了重复的“`syntaxerror`”输出。
3. **解决歧义：**添加 `error` 规则可能会引入新的语法歧义。例如，对于连续的逗号，

分析器可能难以区分是“一个多余的逗号”还是“两个多余的逗号”。本次实验通过精心安排规则的顺序来解决这个问题：Bison/Yacc 会优先匹配更长、更具体的规则。将 Values COMMA COMMA error 放在 Values COMMA error 之前，可以确保 fail05.json("[,]")被正确识别为“double extra comma”。

三、实验环境

容器化工具：Docker

基础镜像：ubuntu:22.04

云环境：由于本地 Docker 网络问题，本次实验借助云端环境 GitHub Codespaces 完成 Docker 镜像的构建与打包。

核心工具：build-essential(提供 gcc 等),flex,bison

测试语言：python3

四、实验内容

本次实验要求为 JSON 解析器添加详细的错误处理和恢复功能。实验提供了一个 syntax.y 骨架和 jsonparser_test.py 测试脚本。核心任务是修改 syntax.y，定义一系列包含 error 标记的语法规则，使得解析器在遇到不符合 JSON 规范的输入文件时，能够捕获这些错误、打印出对应的提示信息，并尽可能地继续解析，最终通过所有 15 个错误测试用例。

五、实验过程与结果分析

1. 环境配置

使用 GitHub 的 Codespaces 辅助配置 docker 环境。

Dockerfile:

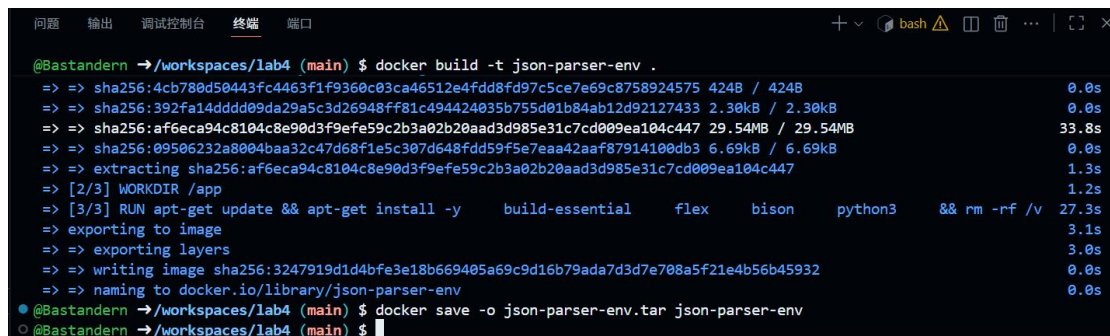
```
FROM ubuntu:22.04
WORKDIR /app
RUN apt-get update && apt-get install -y \
    build-essential \
    flex \
```

af334d0e5bb7b543090c0922e7af44654b94be08a5134f7ece8c27ec1384dd11

```
bison \
python3 \
&& rm -rf /var/lib/apt/lists/*
COPY ..
CMD ["bash"]
```

利用 GitHub Codespaces 功能，在云端环境中成功构建了 Docker 镜像 json-parser-env。将云端构建好的镜像打包为“json-parser-env.tar”文件并下载至本地。

```
docker build -t json-parser-env .           # 构建镜像
docker save -o json-parser-env.tar json-parser-env #打包镜像
```



```
@Bastandern →/workspaces/lab4 (main) $ docker build -t json-parser-env .
=> => sha256:4cb780d50443fc4463f1f9360c03ca46512e4fdd8fd97c5ce7e69c8758924575 424B / 424B 0.0s
=> => sha256:392fa14dd09da29a5c3d26948ff81c494424035b755d01b84ab12d92127433 2.30kB / 2.30kB 0.0s
=> => sha256:af6eca94c8104c8e90d3f9efe59c2b3a02b20aad3d985e31c7cd009ea104c447 29.54MB / 29.54MB 33.8s
=> => sha256:09506232a8004baa32c47d68f1e5c307d648fdd59f5e7eaa42aaf87914100db3 6.69kB / 6.69kB 0.0s
=> => sha256:af6eca94c8104c8e90d3f9efe59c2b3a02b20aad3d985e31c7cd009ea104c447 1.3s
=> [2/3] WORKDIR /app 1.2s
=> [3/3] RUN apt-get update && apt-get install -y build-essential flex bison python3 && rm -rf /v 27.3s
=> exporting to image 3.1s
=> exporting layers 3.0s
=> writing image sha256:3247919d1d4bfe3e18b669405a69c9d16b79ada7d3d7e708a5f21e4b56b45932 0.0s
=> naming to docker.io/library/json-parser-env 0.0s
@Bastandern →/workspaces/lab4 (main) $ docker save -o json-parser-env.tar json-parser-env
@Bastandern →/workspaces/lab4 (main) $
```

下载该文件到本地后，使用 docker load 将镜像加载到本地 Docker 中：

```
docker load -i json-parser-env.tar
```

```
C:\Users\62477\Desktop\22\作业\lab4\lab4>docker load -i json-parser-env.tar
fa015e6a5c38: Loading layer [=====>] 1.536kB/1.536kB
fc9ce94f5d2f: Loading layer [=====>] 308.6MB/308.6MB
Loaded image: json-parser-env:latest
```

最后，启动容器并挂载工作目录,成功搭建了实验环境。

```
docker run -it --rm -v "$(pwd):/app" json-parser-env
```

```
PS C:\Users\62477\Desktop\22\作业\lab4\lab4\BUPT-Compiler-Lab4-2025Fall-master\jp> docker run -it --rm -v "$(pwd):/app"
json-parser-env
root@af334d0e5bb7:/app# make jp
```

2. 编译与问题修复

进入容器后，首先尝试编译项目。

```
make jp
```

遇到链接错误：编译失败，提示链接错误。

```
root@af334d0e5bb7:/app# make jp
flex lex.l
bison -t -d syntax.y
gcc syntax.tab.c -lfl -ly -o jp.out
/usr/bin/ld: cannot find -ly: No such file or directory
collect2: error: ld returned 1 exit status
make: *** [Makefile:11: jp] Error 1
```

-ly 是 Yacc 的旧库，Bison 已不再需要。因此，编辑 Makefile，将 jp 目标的编译命令中的 -ly 标志删除。

重新编译成功，生成 jp.out。

3. 核心代码：syntax.y 错误处理规则

根据实验要求，关键代码是 syntax.y 中用于错误恢复的规则。

- (1) **静默 yyerror**：为防止 Bison 默认的“syntax error”与我们自定义的 printf 信息重复，yyerror 被设置为空函数。
- (2) **JSON 值后多余内容(fail07,fail08,fail10)**：在顶层 Json 规则中，增加一条 Valueerror 规则，捕获解析完一个合法 JSON 值后多余的非法内容。
- (3) **未关闭的对象/数组(fail02)**：在 Object 和 Array 规则中，添加在成员/值列表后捕获 error 的规则。
- (4) **括号不匹配(fail33)**：通过定义“错误”的闭合规则，捕获 LC...RB 或 LB...RC 这样的情况。
- (5) **缺失/错误的冒号(fail19,fail20,fail21)**：在 Member 规则中，允许 STRING 和 Value 之间出现 error，代替 COLON。
- (6) **数组/对象中多余的逗号(fail04,fail05,fail09,fail32)**：这是最容易产生冲突的地方。
 - **对象(fail09,fail32)**：在 Members 列表末尾捕获多余的逗号。
 - **数组(fail04,fail05)**：为解决歧义，必须将最具体的规则(COMMA COMMA error)

放在通用规则（COMMA error）之前。

- (7) **数组中缺失值(fail06):** 通过 error COMMA Value 规则捕获在逗号前缺失值的情况。

完整的 syntax.y 如下:

```
%{
    #include "lex.yy.c"
    #include <stdio.h>
    #include <stdlib.h>

    void yyerror(const char*);
    extern FILE* yyin;
}%

%token LC RC LB RB COLON COMMA
%token STRING NUMBER
%token TRUE FALSE VNULL

%%

Json:
    Value
    /* 捕获 JSON 值结束后多余的内容 (fail10, 07, 08) */
    | Value error { printf("syntax error: Extra value after close, recovered\n"); }
    ;

Value:
    Object
    | Array
    | STRING
    | NUMBER
    | TRUE
    | FALSE
    | VNULL
    ;

Object:
    LC RC
    | LC Members RC
    /* 捕获未关闭的对象 (fail02) */
    | LC Members error { printf("syntax error: Unclosed object, recovered\n"); }
```

```
| LC error { printf("syntax error: Unclosed object, recovered\n"); }
/* 捕获不匹配的括号 (fail33) */
| LC Members RB { printf("syntax error: mismatch, recovered\n"); }
| LC RB { printf("syntax error: mismatch, recovered\n"); }
;
```

Members:

```
Member
| Members COMMA Member
/* 捕获对象末尾多余的逗号 (fail32, 09) */
| Members COMMA error { printf("syntax error: Comma instead if closing
brace/Extra comma, recovered\n"); }

/* 捕获 fail13 (013), 这会被 lexer 识别为 NUMBER(0) NUMBER(13) */
| Members Value { printf("syntax error: Numbers cannot have leading zeroes,
recovered\n"); }
;
```

Member:

```
STRING COLON Value
/* 捕获键值对之间缺失或错误的冒号 (fail19, 20, 21) */
| STRING error Value { printf("syntax error: Missing/Wrong colon, recovered\n"); }
;
```

Array:

```
LB RB
| LB Values RB
/* 捕获未关闭的数组 (fail02) */
| LB Values error { printf("syntax error: Unclosed array, recovered\n"); }
| LB error { printf("syntax error: Unclosed array, recovered\n"); }
/* 捕获不匹配的括号 (fail33) */
| LB Values RC { printf("syntax error: mismatch, recovered\n"); }
| LB RC { printf("syntax error: mismatch, recovered\n"); }
;
```

Values:

```
Value
| Values COMMA Value
/* 捕获数组中用冒号代替逗号 (fail22) */
| Values COLON Value { printf("syntax error: Colon instead of comma,
recovered\n"); }
/* 捕获数组开头或元素前缺失值 (fail06) */
| error COMMA Value { printf("syntax error: <-- missing value, recovered\n"); }
```

```
/* 匹配 fail05 (...,...), 只打印一行 */
| Values COMMA COMMA error { printf("syntax error: double extra comma,
recovered\n"); }

/* 匹配 fail04 (...,...), 只打印一行 */
| Values COMMA error { printf("syntax error: extra comma, recovered\n"); }
;
%%

/* 错误报告函数保持安静, 防止打印多余的 "syntax error:" */
void yyerror(const char *s){
    /* Be silent */
}

/* 主函数, 用于打开文件并启动解析器 */
int main(int argc, char **argv){
    if(argc != 2) {
        fprintf(stderr, "Usage: %s <file_path>\n", argv[0]);
        exit(-1);
    }
    else if(!(yyin = fopen(argv[1], "r"))) {
        perror(argv[1]);
        exit(-1);
    }
    yyparse();
    return 0;
}
```

4. 运行测试

完成 syntax.y 的修改并编译后, 运行 Python 测试脚本。

```

root@af334d0e5bb7:/app# python3 jsonparser_test.py
For file fail02.json:
-----
["Unclosed array"]
-----
syntax error: Unclosed array, recovered
#####
For file fail04.json:
-----
["extra comma",]
-----
syntax error: extra comma, recovered
#####
For file fail05.json:
-----
["double extra comma",,,]
-----
syntax error: double extra comma, recovered
#####
For file fail06.json:
-----
[ , "<-- missing value"]
-----
syntax error: <-- missing value, recovered
#####
For file fail07.json:
-----
["Comma after the close"],
-----
syntax error: Extra value after close, recovered
#####
For file fail08.json:
-----
["Extra close"]]
-----
syntax error: Extra value after close, recovered
#####

```



```

For file fail09.json:
-----
{"Extra comma": true,}
-----
syntax error: Comma instead if closing brace/Extra comma, recovered
#####
For file fail10.json:
-----
{"Extra value after close": true} "misplaced quoted value"
-----
syntax error: Extra value after close, recovered
#####
For file fail13.json:
-----
{"Numbers cannot have leading zeroes": 013}
-----
syntax error: Numbers cannot have leading zeroes, recovered
#####
For file fail19.json:
-----
{"Missing colon" null}
-----
syntax error: Missing/Wrong colon, recovered
#####
For file fail20.json:
-----
{"Double colon": : null}
-----
syntax error: Missing/Wrong colon, recovered
#####
For file fail21.json:
-----
{"Comma instead of colon", null}
-----
syntax error: Missing/Wrong colon, recovered
#####
For file fail22.json:
-----
["Colon instead of comma": false]
-----
syntax error: Colon instead of comma, recovered
#####

For file fail32.json:
-----
{"Comma instead if closing brace": true,
-----
syntax error: Comma instead if closing brace/Extra comma, recovered
#####
For file fail33.json:
-----
["mismatch"}
-----
syntax error: mismatch, recovered
#####
Recovered/Total: 15/15
root@af334d0e5bb7:/app#

```

结果显示 Recovered/Total:15/15，所有测试用例均按预期通过。

六、问题与解决

1. docker 网络问题

在本地直接构建 Docker 镜像时遇到网络超时错误。通过 GitHub Codespaces 拉取镜像再下载到本地创建 docker 容器即可。

```
C:\Users\62477\Desktop\22\作业\lab4\lab4\BUPT-Compiler-Lab4-2025Fall-master\jp>docker build -t json-parser-env .
[+] Building 46.9s (4/4) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 245B
=> ERROR [internal] load metadata for docker.io/library/ubuntu:22.04
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [auth] library/ubuntu:pull token for registry-1.docker.io
> [internal] load metadata for docker.io/library/ubuntu:22.04:
Dockerfile:1
-----
1 | >>> FROM ubuntu:22.04
2 |
3 | WORKDIR /app
-----
ERROR: failed to build: failed to solve: failed to fetch oauth token: Post "https://auth.docker.io/token": dial tcp [2a03:2880:f12a:83:face:b00c:0:25de]:443: connectex: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/xlulrsej9mb5v8eg6wlq4wdia
```

2. 编译过程中出现链接错误

编译过程中出现链接错误 “cannot find -ly”。

```
root@af334d0e5bb7:/app# make jp
flex lex.l
bison -t -d syntax.y
gcc syntax.tab.c -lfl -ly -o jp.out
/usr/bin/ld: cannot find -ly: No such file or directory
collect2: error: ld returned 1 exit status
make: *** [Makefile:11: jp] Error 1
```

原因分析：Makefile 文件中的链接选项 “-ly” 是 Yacc 时代的产物，用于链接 Yacc 库。而现代的 Bison 已经不再需要这个库了，因此该选项是多余的。

解决方案：编辑 “Makefile” 文件，将 gcc 命令中的 “-ly” 选项删除。

3. 语法规则冲突

现象：添加错误恢复规则可能导致冲突。

分析：在 Values 规则中处理多余逗号时，Values COMMA error 和 Values COMMA COMMA error 存在歧义。

解决：利用 Bison 优先匹配更长、更早定义的规则的特性，将最具体的 Values COMMA COMMA error 写在更通用的 Values COMMA error 之前，从而消除了歧义。

七、总结与心得

通过本次实验，我收获良多。

首先，我深入理解了编译器前端进行错误恢复的必要性和实现原理。一个“健壮”的编译器不应该在遇到第一个错误时就停止，而应该尝试恢复并报告尽可能多的错误。通过使用 Bison 的 error 标记，我学会了如何在语法规则中“预设”错误发生的可能性，并引导分析器跳过错误部分，继续解析。

其次，我掌握了提供友好错误提示的技巧。简单地依赖 yyerror 只能输出泛泛的“syntax error”，而通过在 error 规则的动作中 printf 自定义信息，并“静默”yyerror，可以为用户提供更精确的错误诊断，例如“Unclosed array”或“Missing/Wrong colon”。

最后，我进一步锻炼了工程实践和调试能力。无论是再次面对并熟练解决 Docker 网络问题和-ly 链接错误，还是通过调整规则顺序来解决 error 规则引入的语法冲突，都加深了我对工具链和语法分析器工作机制的理解。