课程项目

燕言言

QQ: 2214871526

邮箱: yanyanthunder@foxmail.com

课程项目

课程项目成绩占比：30%

课程项目

主要内容：实验内容是为一个小型的类C语言（C--，文法参考Appendix A）实现一个编译器。如果你顺利完成了本实验任务，那么不仅你的编程能力将会得到大幅提高，而且你最终会得到一个比较完整的、能将C--源代码转换成MIPS汇编代码的编译器，所得到的汇编代码可以在SPIM Simulator上运行。

实习总共分为五个阶段：词法和语法分析、语义分析、中间代码生成、目标代码生成以及中间代码优化。每个阶段的输出是下一个阶段的输入，后一个阶段总是在前一个阶段的基础上完成。其中，目标代码生成以及中间代码优化均基于第三次中间代码生成。

实验助教：

燕言言　　QQ：2214871526

邮箱：yanyanthunder@foxmail.com

何天行　　QQ：976792132

邮箱：976792132@qq.com

课程项目

实验分组

- 组队模式：
  - **110%（1人）**
  - **100-105%（2人）**
  - **90-95%（3人）**
- 实验内容：
  - 预分配（普通版）
  - 完成必做+选做（基础班）
- 实验成绩构成
  - 按时完成（**20%**）
  - 合理（**20%**）
  - 结果正确（**60%**）
  - 未事先得到许可（一般实验截止**24h**前说明迟交原因）迟交或者抄袭，**0**分

提纲

- 实验一
  - 实验内容
  - 实验平台
  - 环境配置
  - 实验设计
  - 注意事项

实验内容

实验内容

- 实现词法分析和语法分析器的基本功能
  - 识别未定义的字符和不符合词法单元定义的字符
  - 输出词法和语法错误类型和信息
  - 按照语法树遍历打印结点信息
- 实现词法分析和语法分析器的高级功能（选做）
  - 识别八进制数和十六进制数、指数形式的浮点数
  - 识别“//”和“/*…*/”形式的注释

输入格式

- 程序的输入是一个包含C--源代码的文本文件，且需要接收一个输入文件名作为参数
  - 假设程序名为cc、输入文件名为test1、程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1即可获得相应的输出结果

实验内容

错误类型

- 词法错误（错误类型A）
  - 即出现C--词法中未定义的字符以及任何不符合C--词法单元定义的字符，如标识符命名错误
- 语法错误（错误类型B）
  - 即出现Appendix A中未定义的语法规则，如全局变量初始化或者数组下标索引是浮点数

输出格式

实验一要求通过标准输出打印程序的运行结果。对于那些包含词法或者语法错误的输入文件，只要输出相关的词法或语法有误的信息即可。在这种情况下，注意不要输出任何与语法树有关的内容。要求输出的信息包括错误类型、出错的行号以及说明文字，其格式为：

```
Error type [错误类型] at Line [行号]: [说明文字].
```

说明文字的内容没有具体要求，但是错误类型和出错的行号一定要正确，因为这是判断输出的错误提示信息是否正确的唯一标准。

实验平台

OJ平台相关问题

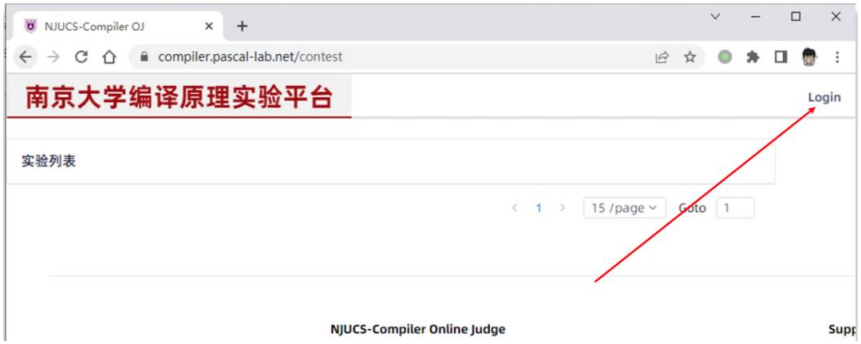
- 用户注册
  - 我们为大家手工注册了**OJ**账号，然后通过邮箱将账号相关信息发送给大家。如果有同学没有收到已经注册的账号或者账号无法登陆，大家可以**QQ**群里联系我们
- 平台输出
  - 平台会以大家的最后一次提交为准
  - 为了更方便大家调试程序，我们也公布了代码的测试结果，包括题目号，输出结果，最终评分等
- 测试用例
  - **OJ**上每一次实验都会添加一定数量的测试用例。大家也可以根据实验内容构造合法的测试用例，测试自己的编译器功能正确性

Vmware 16 Pro版本：http://www.winwin7.com/soft/17946.html#xiazai

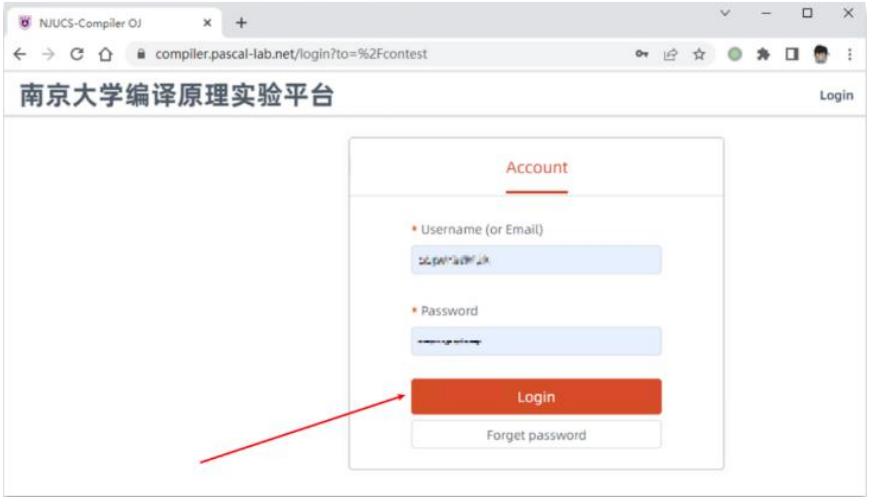
实验平台

OJ登录

（1）打开网站 https://compiler.pascal-lab.net/，点击右上角的“Login”。



（2）填入账号密码，点击“Login”。

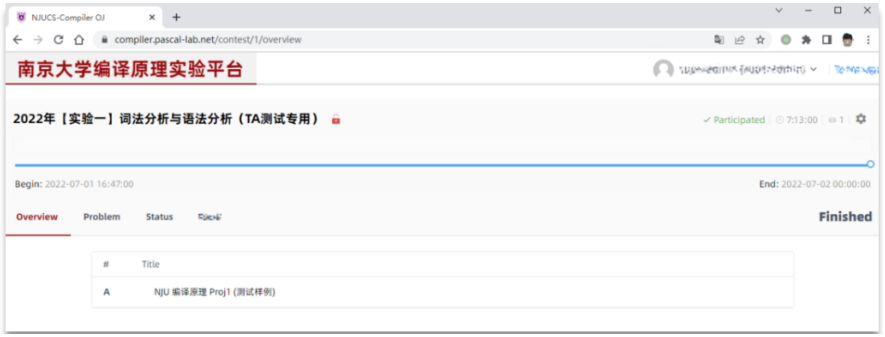


实验平台

OJ实验选择

- 同学们登录平台后，在网站主页“实验列表”下选择一个要进入的实验，点击进入实验详情页
  - 实验详情页内有三个**Tab**页，分别为“**Overview**”、“**Problem**”、“**Status**”

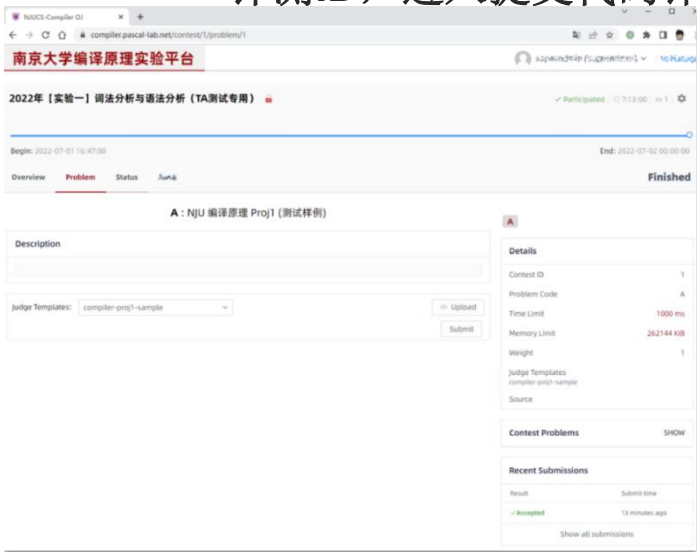
（1）“Overview”显示该实验中设置的题目数量及其基本信息（题号、名称）；点击具体的题目，会跳到该题的“Problem”页。

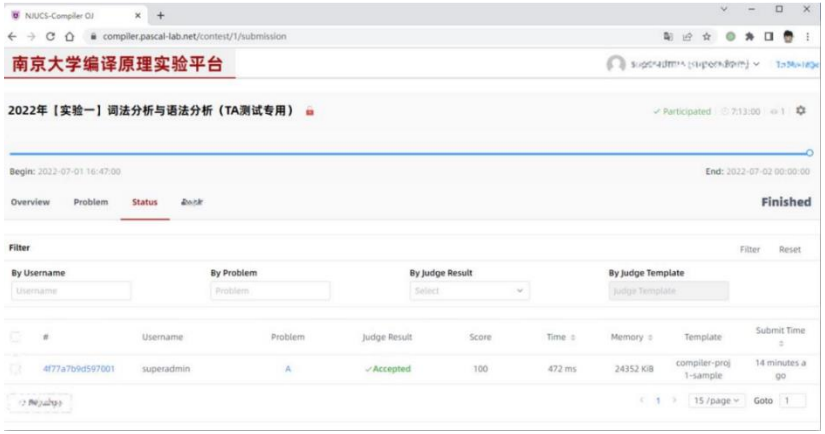


实验平台

OJ实验详情

- “Problem”是题目的详情页，包含：
  - “Description”（详细介绍）、“Details”（基本配置，如时空限制）、“Recent Submissions”（最近提交代码情况）、题目切换以及评测等
  - “Status”是提交码列表页，包含同学们提交的代码及代其评测结果，点击评测ID，进入提交代码详情页





实验平台

OJ代码提交

- 同学们上传的实验代码需要打包成ZIP包

```
张三_1234567.zip
├── Code
│   ├── lexical.l
│   ├── main.c
│   ├── Makefile
│   └── ...
├── README
└── report.pdf
```

正确的压缩包结构

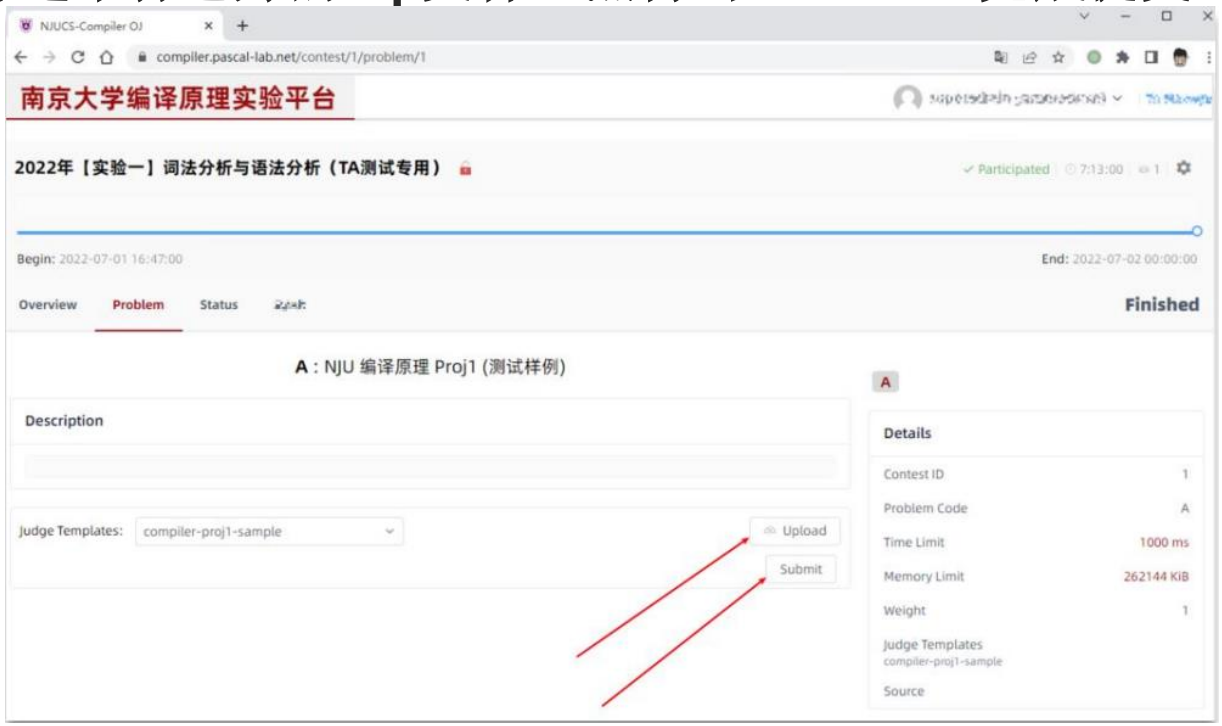
```
# 以下是错误的 Zip 文件结构
张三_1234567.zip
|
张三_1234567
├── Code
│   ├── lexical.l
│   ├── main.c
│   ├── Makefile
│   └── ...
├── README
└── report.pdf
```

不正确的压缩包结构

实验平台

OJ代码提交

- 选择对应的实验，进入“**Problem**”详情页后，点击“**Upload**”并选中打包好的**Zip**文件，点再击**Submit**“完成提交。”



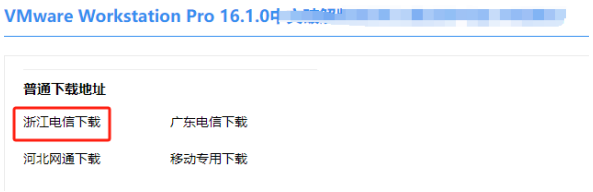
环境配置

- 实验环境
  - GNU Linux Release: Ubuntu 20.04, kernel version 5.13.0-44-generic
  - GCC version 7.5.0
  - GNU Flex version 2.6.4
  - GNU Bison version 3.5.1
- Windows下环境搭建
  - 推荐使用VMware[1] + Ubuntu 20.04 desktop映像文件[2]
  - VMware上安装Ubuntu20.04教程[3]，可以跳过前面的Ubuntu映像文件下载，使用学校的资源。VMware Tools安装参考[4]

[1]. Vmware 16 Pro版本：http://www.winwin7.com/soft/17946.html#xiazai
[2]. Ubuntu 20.04 desktop版本：https://mirrors.nju.edu.cn/ubuntu-releases/20.04/
[3]. 安装教程：https://blog.csdn.net/jiu_liu/article/details/121597146
[4]. VMware Tools安装：https://blog.csdn.net/dengjin20104042056/article/details/106396644

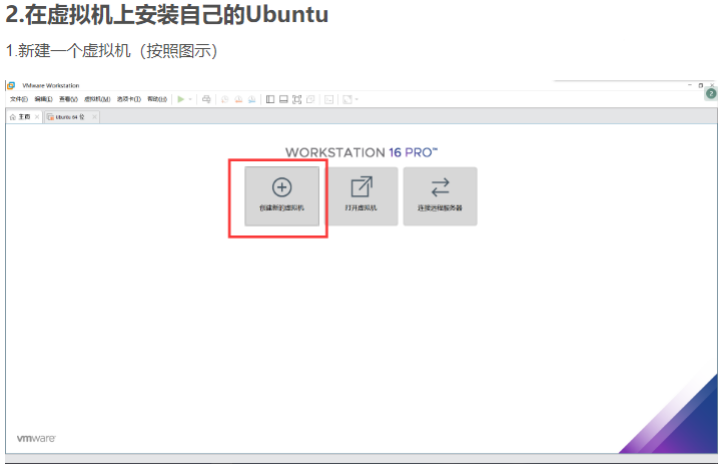
环境配置

- **Windows下环境搭建**
  - **推荐使用VMware[1] + Ubuntu 20.04 desktop映像文件[2]**
  - **VMware上安装Ubuntu20.04教程[3]，可以跳过前面的Ubuntu映像文件下载，使用学校的资源**



Vmware Pro 16.1.0下载位置

Ubuntu 20.04下载位置



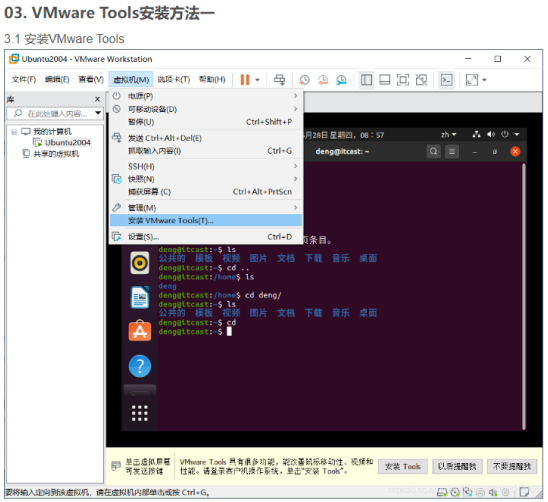
安装起始位置

[1]. Vmware 16 Pro版本：http://www.winwin7.com/soft/17946.html#xiazai
[2]. Ubuntu 20.04 desktop版本：https://mirrors.nju.edu.cn/ubuntu-releases/20.04/
[3]. 安装教程：https://blog.csdn.net/jiu_liu/article/details/121597146

环境配置

Windows下环境搭建

- **VMware Tools [1]** 安装之后，可以实现文件在虚拟机和主机之间传递。其他文件传递方式还有建立共享文件夹或者使用网盘等



从CD-ROM安装VMware Tools

工具安装包

命令行安装

[1]. VMware Tools安装：https://blog.csdn.net/dengjin20104042056/article/details/106396644

环境配置

- **Ubuntu源更新**
  - Ubuntu默认的源较慢，可以替换成学校的源[1]
  - 源替换之后，运行以下命令，完成源更新：
    - sudo apt update
    - sudo apt upgrade
- **GCC安装**
  - Ubuntu20.04默认的gcc版本9.3.0，我们实验默认的是7.5.0
    - sudo apt install gcc-7 g++-7
    - sudo ln -s /usr/bin/gcc-7 /usr/bin/gcc
  - 安装之后，使用ln命令为gcc-7建立软链接，就可以在终端使用gcc了

[1]. Ubuntu源更新：https://nju-mirror-help.njuer.org/ubuntu.html
[2]. Gcc安装：https://blog.csdn.net/ggggyj/article/details/117691948

环境配置

Ubuntu源更新

- Ubuntu默认的源较慢，可以替换成学校的源[1]
- 源替换之后，运行以下命令，完成源更新：
  - sudo apt update
  - sudo apt upgrade

sources内容

[1]. Ubuntu源更新：https://nju-mirror-help.njuer.org/ubuntu.html

环境配置

- **Flex、Bison安装**
  - 在Ubuntu终端使用如下命令安装flex和bison
    - sudo apt install flex
    - sudo apt install bison
  - 安装后，可以通过如下命令查看flex和bison版本
    - flex --version
    - bison --version
- **Make安装**
  - 本学期实验脚本都使用makefile编译链接，因此需要再安装make
    - sudo apt install make

[1]. Ubuntu源更新：https://nju-mirror-help.njuer.org/ubuntu.html
[2]. Ubuntu 20.04 desktop版本：https://releases.ubuntu.com/20.04.6/

实验设计

Flex使用

- Flex的相关资料可以参考实践技术GNU Flex介绍，Flex源代码包含三部分：定义、规则和用户自定义代码。以图示代码为例，实现其词法分析程序步骤如下：
  - 创建Flex文件处理词法分析，如lexical.l
  - 实现定义部分：包含头文件，书写正则表达式

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
%{
    #include<stdio.h>
    #include<string.h>
    int yycolumn = 1;
    int oct_to_dec(char* text);
%}
letter    [A-Za-z]
digit     [0-9]
INT       0|([1-9]{digit}*)|([0][1-7]+)
FLOAT     ({INT}[.]{digit}+)
ID        ([_A-Za-z])({letter}|{digit}|_)*
RELOP     ">"|"<"
delimiter [ \t\r]
EOF       <<EOF>>
%option yylineno
```

Flex定义部分

实验设计

Flex使用

- Flex源代码包含三部分：定义、规则和用户自定义代码。以图示代码为例，实现其词法分析程序步骤如下：
  - 实现规则部分：终结符、标识符和整数识别

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
";"         {printf(";");}
","         {printf(",");}
"="         {printf("=");}
"+"         {printf("+");}
"("         {printf("(");}
")"         {printf(")");}
"{"         {printf("{");}
"}"         {printf("}");}
{ID}        {
                printf("%s", yytext);
            }
{INT}    {
                if ('0' == yytext[0]){
                    printf("%d", oct_to_dec(yytext));
                }
                else {
                    printf("%d", atoi(yytext));
                }
            }
```

Flex规则部分

实验设计

Flex使用

- Flex源代码包含三部分：定义、规则和用户自定义代码。以图示代码为例，实现其词法分析程序步骤如下：
  - 实现规则部分：浮点数、换行符、分隔符以及词法错误识别（词法错误提示信息要重定向到stderr）

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
{FLOAT}    {
                printf("%lf", atof(yytext));
           }
\n         {
                yycolumn = 1;
                printf("\n");
           }
{delimiter} {
              printf("%s", yytext);
            }
.          {
                fprintf(stderr, "Error type A at line %d: Mysterious character \"%s\"\n",
 yylineno, yytext);
           }
```

Flex规则部分

实验设计

Flex使用

- Flex源代码包含三部分：定义、规则和用户自定义代码。以图示代码为例，实现其词法分析程序步骤如下：
  - 用户自定义函数部分：八进制转十进制函数定义

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
int oct_to_dec(char *text)
{
    int len = strlen(text);
    int decimal = 0;
    int ch;
    for(int i=1; i < len; i++){
        ch = (int)text[i] - 48;
        decimal = (decimal + ch) * 8;
    }
    decimal = decimal / 8;
    return decimal;
}
```

Flex规则部分

实验设计

Flex使用

- Flex源代码包含三部分：定义、规则和用户自定义代码。以图示代码为例，实现其词法分析程序步骤如下：
  - 用户自定义代码部分，实现为**main.c**，以读取解析**C—**源码

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
1  #include<stdio.h>
2  extern FILE* yyin;
3  extern char* yylex();
4  int main(int argc, char** argv) {
5      if (argc > 1) {
6          if (!(yyin = fopen(argv[1], "r"))) {
7              perror(argv[1]);
8              return 1;
9          }
10     }
11     // Lexical Analysis
12     while (yylex() != 0);
13     return 0;
14 }
```

main.c

实验设计

Flex使用

- Flex源代码包含三部分：定义、规则和用户自定义代码。编译链接生成scanner文件，扫描c—源码后，结果如下：
  - Scanner准确识别出标识符、数字、符号、词法错误

```
1  int main()
2  {
3      int i = 11222;
4      int j = ~i;
5      int k = 012;
6      float l = 2.3;
7  }
```

C—源码

```
lulu@ubuntu:~/lab1/lab1/Sample1$ flex lexical.l
lulu@ubuntu:~/lab1/lab1/Sample1$ gcc main.c lex.yy.c -lfl -o scanner
lulu@ubuntu:~/lab1/lab1/Sample1$ ./scanner sample1.cmm
int main()
{
    int i = 11222;
    int j = Error type A at line 4: Mysterious character "~"
i;
    int k = 10;
    float l = 2.300000;
}
```

词法分析结果

实验设计

Bison使用

- 创建Bison文件处理语法分析，如parser.y
- 如果-ly报错，可以使用如下命令：
  - sudo apt install libbison-dev
- 改进flex源文件
  - 定义扩展，加入行号信息

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

C—源码

```
%{
    #include "syntax_arr.tab.h"
    int yycolumn = 1;
    #define YY_USER_ACTION \
        yylloc.first_line = yylloc.last_line = yylineno; \
            yylloc.first_column = yycolumn; \
            yylloc.last_column = yycolumn + yyleng - 1; \
            yycolumn += yyleng;
    extern void printError(char errorType, int lineno, char* msg);
    extern int Err_new(int errorLineno);
    extern int errorflag;
    extern int oct_to_dec(char* s);
%}
letter      [A-Za-z]
digit       [0-9]
INT         0|([1-9]{digit}*)|([0][0-7]+)
FLOAT       ({INT}[.]{digit}+)
ID          ([_A-Za-z])({letter}|{digit}|_)*
RELOP       ">"|"<"
delimiter   [ \t\r]
EOF         <<EOF>>
%option yylineno
%%
```

Flex定义部分扩展

实验设计

Bison使用

- 改进flex源文件
  - 返回语法符号

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

C—源码

```
{RELOP} {return RELOP; }
";"     {return SEMI; }
","     {return COMMA; }
"="     {return ASSIGN; }
"+"     {return PLUS; }
"-"     {return MINUS; }
"."     {return DOT; }
"("     {return LP; }
")"     {return RP; }
"["     {return LB; }
"]"     {return RB; }
"{"     {return LC; }
"}"     {return RC; }
```

Flex规则终结符

```
{INT}    {   if('0' == yytext[0]){
                yylval.type_int = oct_to_dec(yytext);
             }else{
                yylval.type_int = atoi(yytext);
             };
             return INT;
         }
{FLOAT} {
             yylval.type_float = atof(yytext);
             return FLOAT;
         }
"int"|"float"   {
                     yylval.type_string = strdup(yytext);
                     return TYPE;
                 }
"//"     {
             char c = input();
             while (c != '\n') c = input();
         }
{ID}     {
             yylval.type_string = strdup(yytext);
             return ID;
         }
\n       { yycolumn = 1; }
{delimiter}    { }
.        {
             if (Err_new(yylineno)) {
                 char msg[32];
                 sprintf(msg, "Mysterious character \"%s\"", yytext);
                 printError('A', yylineno, msg);
```

Flex规则非终结符处理

实验设计

Bison使用

Bison源文件定义

- 定义部分

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

C—源码

```
%locations
%{
    #include <stdio.h>
    #include <string.h>
    #include <stdarg.h>
    #include <ctype.h>
    #include "lex.yy.c"

    void yyerror(const char* s);
    int errorflag = 0;
    int last_error = 0;
    enum NodeType {
        NTML,
        NVL,
        VL
    };
    struct Node {
        char* nodeName;
        enum NodeType nodeType;
        int lineNum;
        union {
            int Valint;
            float Valfloat;
            char* Valstr;
        };
        struct Node* firstChild;
        struct Node* Sibc;
    };
```

Bison定义实例

```
%token RELOP ASSIGN
%token SEMI COMMA
%token PLUS MINUS
%token DOT
%token LP RP LB RB LC RC
%token <type_int> INT
%token <type_float> FLOAT
%token <type_string> ID TYPE

%type <type_pnode> Program ExtDefList ExtDef ExtDecList Specifier
%type <type_pnode> VarDec FunDec VarList ParamDec CompSt
%type <type_pnode> StmtList Stmt DefList Def DecList Dec Exp

%nonassoc error
%right ASSIGN
%left RELOP
%left PLUS MINUS
%left LP RP LB RB DOT
```

Bison符号及优先级

实验设计

Bison使用

Bison源文件定义

- 规则部分

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

```
Program : ExtDefList {
            $$ = constructNode("Program", NTML, @$.first_line);
            construct($$, 1, $1);
            Root = $$;
        }
    | ExtDefList error {
            if (Err_new(@2.first_line)) {
                printError('B', @2.first_line, "Unexpected character");
                struct Node* nodeError = constructNode("error", NVL, @2.first_line);
                $$ = constructNode("Program", NTML, @$.first_line);
                construct($$, 2, $1, nodeError);
                Root = $$;
            } else {
                $$ = NULL;
            }
        }
    ;
```

C—源码　　Bison产生式实现

Bison使用

Bison源文件定义

- 数组处理

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

```
VarDec : ID {
            struct Node* nodeID = constructNode("ID", VL, @1.first_line);
            nodeID->Valstr = $1;
            $$ = constructNode("VarDec", NTML, @$.first_line);
            construct($$, 1, nodeID);
        }
    | VarDec LB INT RB {
            struct Node* nodeINT = constructNode("INT", VL, @3.first_line);
            nodeINT->Valint = $3;
            $$ = constructNode("VarDec", NTML, @$.first_line);
            construct($$, 4, $1, constructNode("LB", NVL, @2.first_line), nodeINT, const
ructNode("RB", NVL, @4.first_line));
        }
    | VarDec LB error RB {
            if (Err_new(@3.first_line)) {
                printError('B', @3.first_line, "Syntax error between \"[]\"");
                struct Node* nodeError = constructNode("error", NVL, @3.first_line);
                $$ = constructNode("VarDec", NTML, @$.first_line);
                construct($$, 4, $1, constructNode("LB", NVL, @2.first_line), nodeError,
constructNode("RB", NVL, @4.first_line));
            } else {
                $$ = NULL;
            }
        }
    ;
```

C—源码

Bison数组下标检查

实验设计

Bison使用

- Bison源文件定义
  - 用户函数定义：语法树节点构造

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

```c
struct Node* constructNode(char* Name, enum NodeType Type, int line) {
    //struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* newNode = &node_list[node_idx++];
    newNode->nodeName = Name;
    newNode->nodeType = Type;
    newNode->lineNum = line;
    newNode->firstChild = NULL;
    newNode->Sibc = NULL;
    return newNode;
}
```

C—源码　　节点构造

实验设计

Bison使用

Bison源文件定义

- 用户函数定义：语法树打印

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

C—源码

```
void Print_tree(struct Node* rootNode, int spaceNum) {
    if (rootNode == NULL)
        return;
    for (int i = 0; i < spaceNum; i++) {
        printf(" ");
    }
    switch (rootNode->nodeType) {
        case NTML:
            printf("%s (%d)\n", rootNode->nodeName, rootNode->lineNum);
            break;
        case NVL:
            printf("%s\n", rootNode->nodeName);
            break;
        case VL:
            printf("%s: ", rootNode->nodeName);
            if ((strcmp(rootNode->nodeName, "TYPE") == 0) || (strcmp(rootNode->nodeName,
"ID") == 0)) {
                printf("%s\n", rootNode->Valstr);
            } else if (strcmp(rootNode->nodeName, "INT") == 0) {
                printf("%d\n", rootNode->Valint);
            } else if (strcmp(rootNode->nodeName, "FLOAT") == 0) {
                printf("%f\n", rootNode->Valfloat);
            } else {
                printf("ERROR!");
            }
            break;
        default:
            printf("ERROR!");
```

打印语法数

Bison使用

Bison源文件定义

- 用户函数定义：错误处理

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

```c
void printError(char errorType, int lineno, char* msg) {
    fprintf(stderr, "Error type %c at Line %d: %s.\n", errorType, lineno, msg);
    errorflag = 1;
}

int Err_new(int errorLineno) {
    if (last_error != errorLineno) {
        errorflag = 1;
        last_error = errorLineno;
        return 1;
    } else {
        return 0;
    }
}
```

C—源码　　错误处理

实验设计

- **Bison使用**
  - **Bison mian函数**
    - 用户函数定义：错误处理

```
1  int i;
2  int main() {
3      int a[8][30];
4      a[2,2] = 9;
5      int i =~;
6  }
```

C—源码

```
1  #include<stdio.h>
2  extern int yyparse(void);
3  extern void yyrestart(FILE* input_file);
4  extern int errorflag;
5  extern struct Node* Root;
6  extern void Print_tree(struct Node* rootNode,int spaceNum);
7  extern void Pt_finish(struct Node* rootNode);
8
9  int main(int argc, char** argv) {
10     if (argc > 1) {
11         FILE* f = NULL;
12         if (!(f = fopen(argv[1], "r"))) {
13             perror(argv[1]);
14             return 1;
15         }
16         // Syntax Analysis
17         yyrestart(f);
18         yyparse();
19         if(!errorflag) {
20             Print_tree(Root, 0);
21         }
22         Pt_finish(Root);
23     }
24     return 0;
25 }
```

main函数

注意事项

- 代码相关问题
  - 编译出错
    - 比如**Makefile**编译脚本问题
  - 错误输出过多
    - 错误结果多于测试用例中包含的错误数目
    - 没有输出语法树（前期**OJ**有一些**BUG**，语法树不输出也可以，最近修复了）
  - 输出格式错误
    - 输出重定向
    - 进制解析错误

注意事项

样例

- 编译出错
  - **OJ提供了Makefile脚本，如果大家提供了makefile编译脚本，那么默认情况下make会调用makefile脚本**
  - **OJ要求编译脚本支持编译以及clean，自定义的makefile脚本中要至少包含parser编译，clean目标**

```
1 parser: main.c lexical.l syntax.y
2         flex lexical.l
3         bison -d syntax.y
4         gcc main.c syntax.tab.c -lfl -ly -o parser
5
6 clean:
7         rm -f parser lex.yy.c syntax.tab.c syntax.tab.h syntax.output
8         rm -f *.o
```

makefile脚本定义的基本目标

× Compilation Error

Judge Log

[pass] copy Makefile
[fail] clean and compile, given stdout log:
make: *** No rule to make target 'clean'. Stop.

OJ平台输出

注意事项

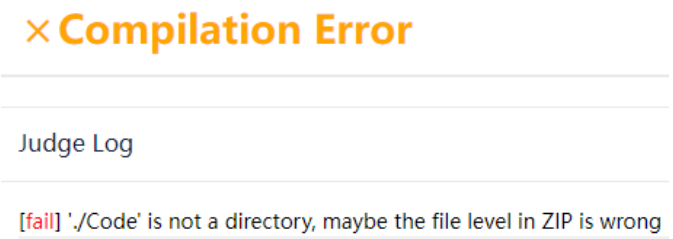
样例

编译出错

- 提交代码格式不符合OJ平台要求，OJ编译出错
- README.md文件必须存在，否则可能无法上传（感谢咱们班的同学，不然我们都没发现这个问题，README.pdf文件也无法上传）

```
张三_1234567.zip
├─── Code
│      ├─── lexical.l
│      ├─── main.c
│      ├─── Makefile
│      └─── ...
├─── README
└─── report.pdf
```

代码目录结构



OJ平台输出

注意事项

- 样例
  - 错误输出过多
    - 错误结果多于测试用例中包含的错误数目
    - 正确的程序没有输出语法树，输出了错误提示信息

```
[!] testing 'E2-2'
[fail] 'E2-2': expected 'Program (1)' at line 1, but given 'Error type A at Line 2: Illegal floating point number ".e1"'
[fail] 'E2-2'
given stdout in test 'E2-2':
Error type A at Line 2: Illegal floating point number ".e1"
```

代码编译出错

注意事项

样例

- 输出格式错误
  - **OJ平台会获取系统stdout输出结果，忽略stderr输出，如果错误报告函数中将输出结果重定向为stderr，OJ可能会忽略部分错误信息内容，从而判断测试用例执行结果不符合预期**
  - 输出内容不符合预期，比如进制换算出错

```
[!] testing 'D-1'
[fail] 'D-1': expected '              INT: 52' at line 26, but given '              INT: 0x34'
[fail] 'D-1'
given stdout in test 'D-1':
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
```

代码编译出错

注意事项

文档相关问题

- 文档缺失
  - 文档内容不超过3页，但最好不要为空
- 文档内容
  - 可以介绍自己实现的细节，比如错误处理或者语法规则
  - 自己对词法语法分析过程的理解与感悟
  - 改进或者尝试，比如手写的词法语法分析器

Lab1 实验报告

实验内容

基于GNU Flex和GNU Bison工具实现了基本的词法分析和语法分析程序，可以针对规则简化后的C语言源程序进行此法和语法分析，输出异常信息或在无异常的情况下生成语法树。

除了词法错误（错误类型A）和语法错误（错误类型B）以外，还完成了以下要求：

- 识别八进制和十六进制数；
- 识别指数形式的浮点数，并检测非法的浮点数错误；
- 过滤单行和多行注释。

实验特色

我在词法分析的步骤加入了很多以 `error` 开头的错误词法，可以实现对8进制和16进制数、浮点数、非法变量名的报错：

```
errornum_o 0(8|9){digit}*
errornum_h (0x|0X)([0-9G-Z]+|[0-9g-z]+)
errorid {digits}{ID}
errorfloat \.{digits}|{digits}\.
errorfloat_e {digits}\.{digit}*[Ee][+-]?|{digit}*\.{digits}[Ee][+-]?|\.[Ee][+-]?{digits}
```

还在词法分析的步骤实现了对注释的消除：

文档示例

注意事项

文档相关问题

- 文档内容
  - 可以介绍自己实现的细节，比如错误处理或者语法规则
  - 改进或者尝试，比如手写的词法语法分析器
  - 自己对词法语法分析过程的理解与感悟

语法树相关定义

多叉树节点定义：

```
struct node
{
    int type; //区别词法和语法单元，方便打印时区分处理。
    int name; //词法|语法单元名称对应的编号。根据编号进行打印。
    int line; //所在的行号
    char value[32]; //词法单元的值
    int child_num; //子树数量
    struct node* children[10]; //子树索引
};
```

创建函数、插入函数、打印函数围绕定义进行相应处理即可。

实验感想

1.真的不能再拖延了！有时间一定要先完成各种任务，不要贪玩！

2.上课要好好听，课本要好好看，好好地理解相应的知识。否则就会觉得实验无从下手！

文档示例

谢谢大家