计算机科学
与技术系

实验一 词法分析和语法分析

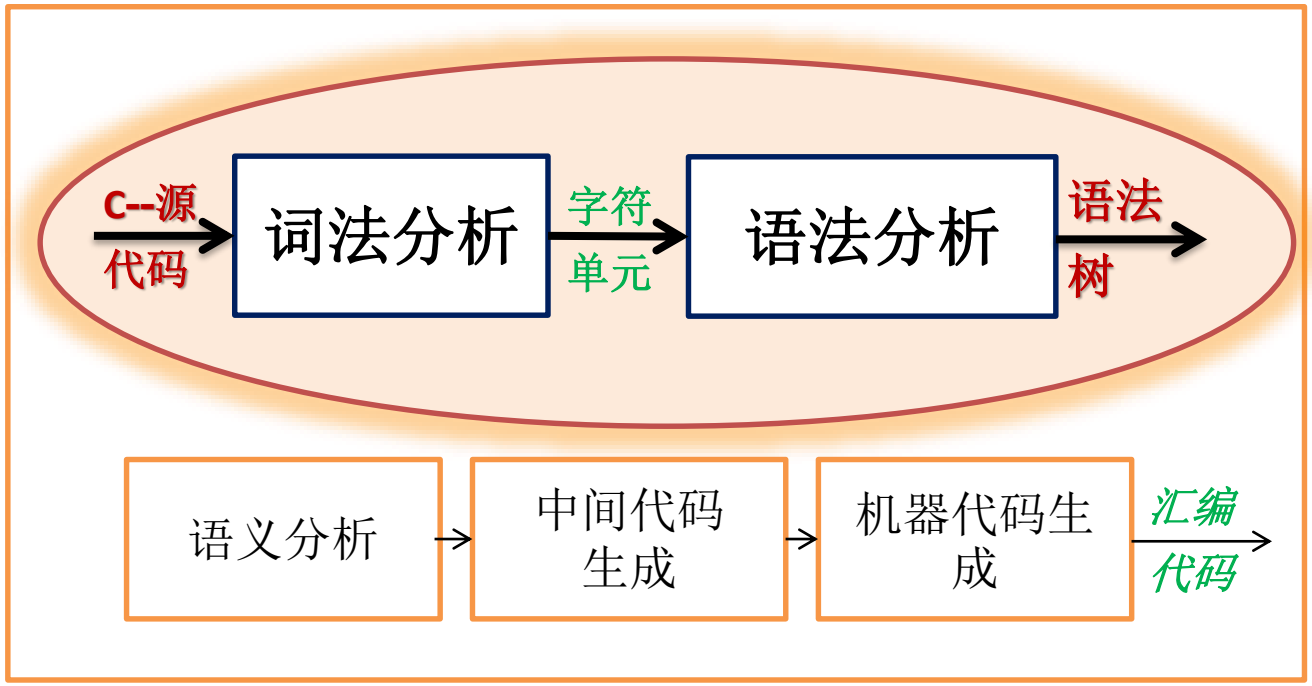
老师：戴新宇，梁红瑾
助教：邱丰羽，欧阳亚文
{qiufy, ouyangyw}@nlp.nju.edu.cn
或者在课程 QQ 群中联系

写在前面

- 实验一与书本上第三、四章的知识对应
- 请首先了解一下附录中的 C-- 语言的文法！
- 请务必花时间认真阅读实验手册，Project_1.pdf，这将是同学们理解整个实验的流程、项目设计、甚至是写代码时的重要参考
- 实验手册的 2.2 节实验指导部分，相当于 Hello World 程序，请按照说明 walk through，就会对项目中的 Flex、Bison 有大体的了解，同时能够建立起一个项目代码的总体结构

概要

- 实验任务
  - 必做部分
  - 选做部分
- 实验讲解
  - 示例
  - 快速上手
- 实验提交
- 实验检查
  - 运行环境
  - 代码查重



实验任务

- 词法分析
  - **必做**：能够查出C--源代码中的词法错误
  - **选做**：识别合法八进制（012✓，081✗）、十六进制整数（0x89/0xAb/0XaB ✓，0xG5 ✗ ），识别指数形式的浮点数（1E-2/01.2E+34/ 43.e-3/.5e02 ✓，9.8E7.6 ✗ ）
- 语法分析
  - **必做**：能够查出C--源代码中的语法错误
  - **选做**：正确处理两种风格的注释
- 没有词法和语法错误的情况，则打印语法树

实验示例 1

- 检测错误类型 1，词法错误

```
1   int main()
2   {
3       int i = 1;
4       int j = ~i;
5   }
```

- 示例输出
  - Error type A at line 4: Mysterious character ‘~’

实验示例 2

- 检测错误类型 2，语法错误

```
1  int main()
2  {
2      float a[10][2];
3      int i;
4      a[5,3] = 1.5;
5      if (a[1][2] == 0)
6          i = 1
7      else
8          i = 0;
9  }
```

1. 代码中可能有多种错误；
2. 这里第二个错误也可以显示在第 7 行，‘;’ is expected but sees else instead.

- 示例输出
  - Error type B at line 4: syntax error
  - Error type B at line 6: syntax error

实验示例 3

- 没有词法、语法错误，打印语法树

```
1  int inc()
2  {
3      int i;
4      i = i + 1;
5  }
```

- 虽然这里没有返回语句...
  - 有语义上错误，但词法、语法没问题
  - 这是实验二的内容 :D

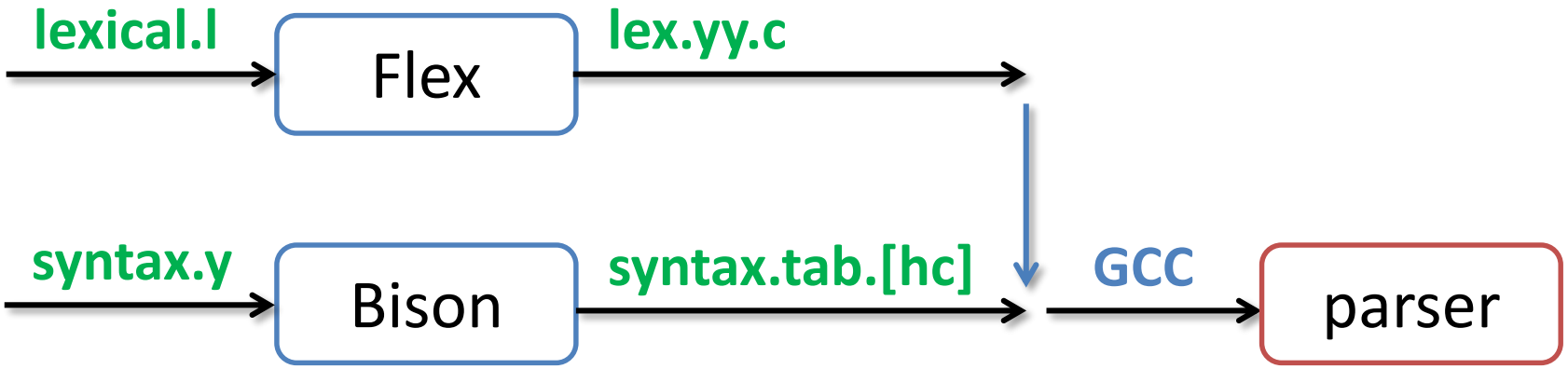
```
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
      FunDec (1)
        ID: inc
        LP
        RP
      CompSt (2)
        LC
        DefList (3)
          Def (3)
            Specifier (3)
              TYPE: int
            DecList (3)
              Dec (3)
                VarDec (3)
                  ID: i
            SEMI
        StmtList (4)
          Stmt (4)
            Exp (4)
              Exp (4)
                ID: i
              ASSIGNOP
              Exp (4)
                Exp (4)
                  ID: i
                PLUS
                Exp (4)
                  INT: 1
            SEMI
        RC
```

实验快速上手 1

- 编译环境及过程
  - GNU Flex, GNU Bison, GCC on Linux Ubuntu
    - `sudo apt-get install flex`
    - `sudo apt-get install bison`
  - 源文件 { lexical.l, syntax.y } → 可执行程序 parser
    - Flex: lexical.l → lex.yy.c
    - Bison: syntax.y → syntax.tab.h, syntax.tab.c
    - GCC: *.c → parser
  - 测试
    - `./parser test.cmm`

实验快速上手 2

- 编译方法



- 编译命令
  - `flex lexical.l`
  - `bison -d syntax.y`
  - `gcc -o parser syntax.tab.c -lfl -ly`
- 我们提供了一个 `Makefile` 作参考，随实验材料发布

实验快速上手 3

- Flex & Bison

```
%{                                  lexical.l
Declarations
#include “ex1.tab.h”
%}
Definitions (RegEx)
%%
Rules
%%
subroutines (e.g main)
```

```
%{                                  syntax.y
Declarations
#include "lex.yy.c"
%}
Definitions (%Token)
%%
Productions
%%
subroutines
```

实验快速上手 4

- Flex: .l 文件格式

```
%{
Declarations
%}
Definitions
%%
Rules       // 将保留字置于标识符{id}之前
%%
subroutines
```

实验快速上手 5

- Flex 程序示例

```
%{
  /* 此处省略#include部分 */
  int chars = 0;
  int words = 0;
  int lines = 0;
%}
letter  [a-zA-Z]
%%
{letter}+  { words++; chars+= yyleng; }
\n  { chars++; lines++; }
.  { chars++; }
%%
int main(int argc, char** argv) {
  if (argc > 1) {
    if (!(yyin = fopen(argv[1], "r"))) {
      perror(argv[1]);
      return 1;
    }
  }
  yylex();
  printf("%8d%8d%8d\n", lines, words, chars);
  return 0;
}
```

运行过程：

```
$ flex test.l
$ gcc  lex.yy.c -lfl -o parser
$ ./parser test.cmm
```

- Flex 有一个内置变量，yytext，类型为 char*，里面保存了当前词法单元 所对应的词素
- yyleng 也是 Flex 提供的变量，这里可以理解为 strlen(yytext)

- 每遇到一个换行符就把行数加一
- 每识别出一个单词就把单词数加一
- 每读入一个字符就把字符数加一
- 最后在main函数中把chars、words和lines的值全部打印出来。

实验快速上手 6

- Flex 中你需要做的内容
  - 我们有
    - yytext, yyleng：词素字符串
    - yylineno：使用 `%option yylineno` 开启，可以记录行号，以便在报错时提示输入文件的哪一行出 现了问题
    - yylval：全局变量，当前词法单元的属性值
  - 在 Flex 的 rules 部分，给出 C-- 词法相关的正则表达式和相应的响应函数，例如
    - 在 definition 部分，有：**`id {letter}({letter}|{digit})*`**
    - 一旦匹配到 id，对应的响应函数可以是
    - `{id} { printf("Line %d: (ID, %s)\n", yylineno, yytext);}`

实验快速上手 7

- Flex 中你需要做的内容
  - 通常我们需要把所有的节点用语法树组织起来
    - 预定义树的结构后，可以有

```
{id} { //printf("Line %d:(ID, %s)\n", yylineno, yytext);
       yylval.node = (TreeNode*)malloc(sizeof(TreeNode));
       yylval.node->lineno = yylineno;
       yylval.node->type = 1;
       yylval.node->tokentype = 26;
       yylval.node->name = malloc(strlen(yytext)+1);
       strcpy(yylval.node->name, yytext);
       return ID;
     }
```

这些都是示例代码，使用需谨慎，并且这不是一种比较好的做法！

实验快速上手 8

- Bison： .y 文件内容

```
%{
Declarations
%}
Definitions
// 考虑优先级与结合性
%%
Productions
%%
subroutines
```

```
%right   ASSIGNOP
%left    AND
%left    RELOP
%left    PLUS MINUS
%left    STAR DIV
%right   NOT UMINUS
%left    DOT LB RB LP RP
```

```
Exp | MINUS EXP %prec UMINUS
Stmt: /* 悬空 else 问题 */
      IF LP Exp RP Stmt %prec LOWER_ELSE
    | IF LP Exp RP Stmt ELSE Stmt
```

实验快速上手 9

- Bison 语法树生成
  - 语法树应该是一棵多叉树
  - 定义树节点数据结构 `struct TreeNode`，大致有：
    - 节点类型：非终结符号、终结符号（数，标识符）
    - 节点名字：Exp，Type，ID 等
    - 所在行号：由 yylineno 给出
    - 某些类型的值，如 struct
    - 数值的属性值：INT，FLOAT 的值
  - 为了方便维护多叉树结构和节点之间的关系，还应该有指向子节点、兄弟节点的指针

实验快速上手 10

- Bison 语法树生成
  - 语法树应该是一棵多叉树
    - 定义树节点数据结构，方便维护父节点、兄弟节点关系

```
Exp: Exp ASSIGNOP Exp {
        $$ = create_node("Exp", @$.first_line, "");
        add_child($$, $1);
        add_sibling($1, $2);
        add_sibling($2, $3);
}
```

    - $$ 是父节点 **Exp**，需要创建一个树节点，这里第一个参数是名字 Exp；第二个参数是这个节点的行号，而第三个参数是这个文法符号对应的字符串
    - add_child, add_sibling 分别将节点作为父节点的子节点以及子节点的兄弟节点添加到多叉语法树中

实验快速上手 11

- Bison 语法树输出
  - 多叉树的输出，符合输出的格式要求
  - 使用层次递归前序遍历即可

```
void printTree(TreeNode *root, int layer) {
    if(root) {
        for(int i = 0; i < layer; ++i)
            printf("  ");
        printf("%s (%d)\n", root->name, root->lineno);
        TreeNode *temp = root->child;
        while(temp) {
            printTree(temp, layer+1);
            temp = temp->sibling;
        }
    }
}
```

实验快速上手 12

- Bison 语法解析的错误恢复产生式
  - Bison 在当前状态对 yylex() 返回的 token 没有定义时即发生了语法错误，调用 yyerror
  - `yyerror(char* str){  printf("syntax error\n"); }`
  - Bison 不断丢弃词法单元直至遇到给出的同步单元（例如通常是分号、括号）
  - 机制：错误恢复产生式
    - 例 Stmt: error SEMI
  - 在语法里指定 error 符号应该放到哪里，需谨慎考虑放置 error 符号的位置，这将最终决定能否检测出各种可能的文法上的错误

实验提交方式

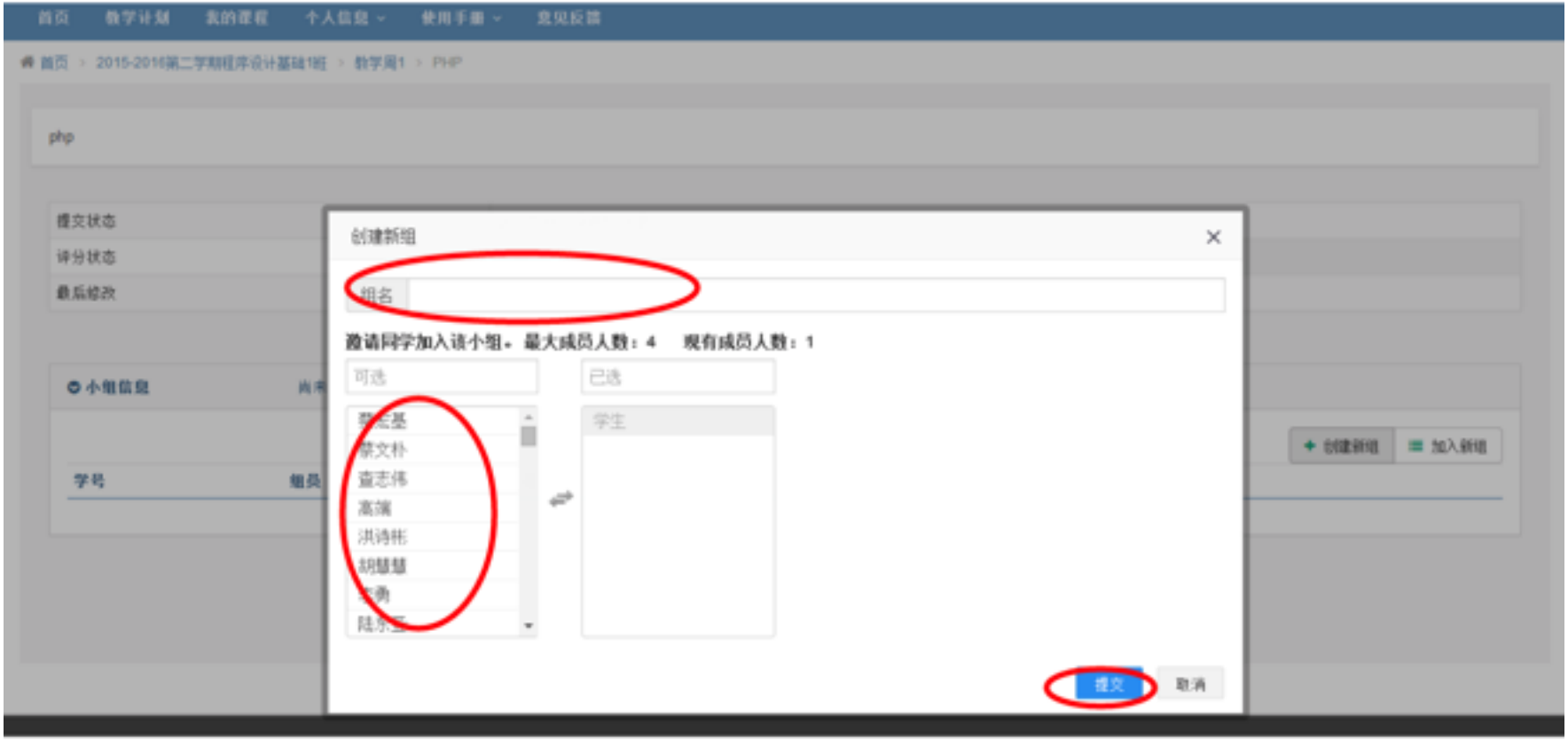
- 提交至：http://cslabcms.nju.edu.cn
  - 使用个人账号登录，独立完成
  - 各位选课同学列表已经导入，请完善个人信息
  - 上述过程中的任何问题请联系助教或老师
  - 无特殊情况不接受其他提交方式
- 提交第四周的 Project 1 词法分析和语法分析项目
  - 所有内容打包并压缩，命名为学号.zip/rar/tar.gz，如161220001.zip
  - 提交后请确认作业状态为“已经提交”、“已提交等待评分”，而不是“草稿(未提交)”
- 截止日期：**10 月 21 日，23:59:59**，请尽量不要在此之前的几分钟提交，网络有风险

创建小组与提交作业

- 点击小组信息，加入一个小组或创建一个小组

- 实验设置为该课程系统中的项目作业，但要求独立完成，所以请创建一个只有自己一个成员的小组后提交

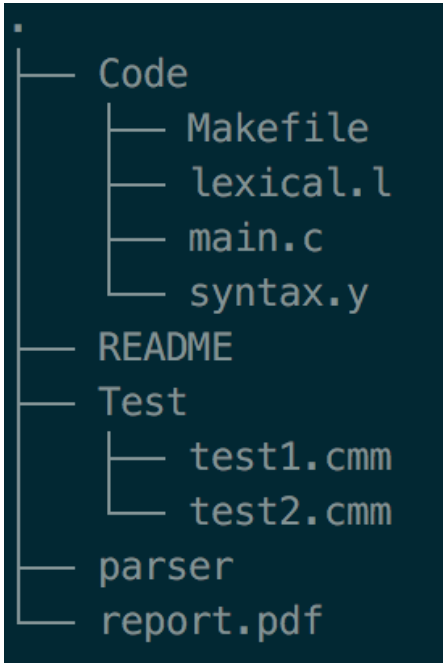
创建小组与提交作业



- 注意：只有在加入某个组之后才能提交作业！

实验提交内容

- 压缩包内容**至少**包含
  - 请遵循右侧的压缩包结构！
  - 源代码（*.l, *.y, 其他 .h, .c 文件）
    - 尽量遵循优秀代码规范，有必要注释
    - 请使用 Makefile，可以自建，即便使用提供的版本，请一并放在代码文件夹内
  - 可执行文件（命名为 parser）
  - 实验报告
    - 推荐 pdf， doc/docx 也可以
    - 报告内容包括个人信息、完成的功能点，实现方法，用到的数据结构表示，编译运行方法，实验总结等
    - **不要贴大段代码，篇幅不超过 4 页**
  - README 可选



实验检查

- 推荐检查环境
  - Ubuntu 18.04 LTS
  - GCC 5.4.x
  - GNU Flex 2.6.0
  - GNU Bison 3.0.4
- 事实上，如果不使用较冷门或非正式、非稳定版本，基本不会有兼容问题，
  - 甚至可以在 macOS 上完成
    - 编译选项中的 `-lfl` 需要改为 `-ll`
  - 黑科技：Flex Bison step by step compile on windows

实验评分

- 你可以先试试实验手册上的测试样例，我们会使用其他的测试样例，根据通过样例情况评分
- **必做部分（80%）**
  - 识别词法（10%）、语法（50%）错误
  - 正确输出没有词法、语法错误代码的语法树（20%）
- **实验报告及代码风格、实现方式等（20%）**
- 选做部分（bonus 15%）
  - 识别八进制、十六进制整数（5%）
  - 识别指数形式（含有E、e）浮点数（5%）
  - 识别两种注释风格（5%）

严格的代码查重

- 参考网上的任何代码请指明出处！
- 区别参考与抄袭，任何形式的代码抄袭都是不允许的！被确认的抄袭者与被抄袭者本次实验都记为 0 分！

- 祝你好运！

写在最后

- 本次实验是本学期实验的第一阶段，包括词法分析和语法分析两部分， 这将是以后完成各阶段实验的基础，具有前后依赖和继承的关系，务必重视！
- **预告**：后续实验是很大程度上是基于本次实验得到的语法树进行的， 请小心地设计语法树中的节点数据结构和树的组织形式。这应该是本次实验中最富有创意的地方了，也是难点
- 实验手册中的讲解已经较为详细了，但如果有任何困难，请联系任何一位助教，我们会帮助你。当然，这不会对你最终实验的得分有任何影响
- 如果想要完成一些高级功能，当然是十分欢迎的！请在实验报告中指出，据此会有一定的奖励加分。如果你没有精力完成选做部分， 没有关系，漂亮地完成基本功能也可以得到满分，并且不会对后续的实验有影响

Thank you~
Q & A