中山大學 SUN YAT-SEN UNIVERSITY

计算机学院（软件学院） SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第1讲：概述(2)

张献伟

xianweiz.github.io

DCS290, 2/29/2024

sysu-编译(张)

群号: 189205980

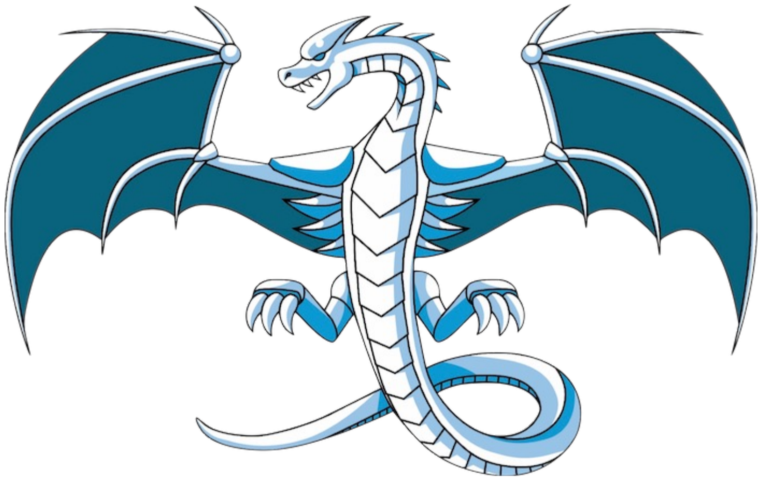
扫一扫二维码，加入群聊

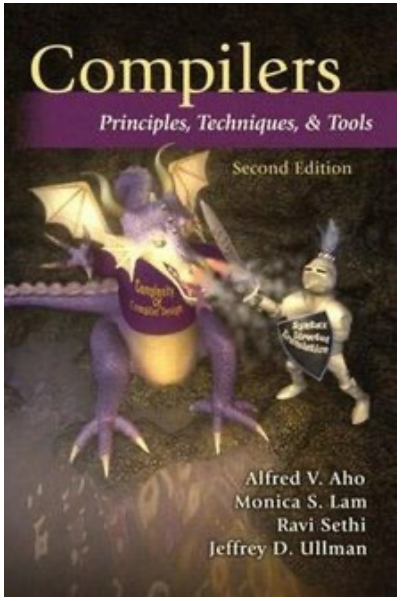
QQ

中山大學 SUN YAT-SEN UNIVERSITY

LLVM Logo

- The LLVM logo is a stylized wyvern (a kind of dragon)
  - Dragons have connotations of power, speed and intelligence, and can also be sleek, elegant, and modular (err, maybe not)
  - In addition, there is a series of influential compiler books going back to 1977 which featured dragons on the cover
  - Wyvern is a type of mythical dragon with two legs, two wings, and often a pointed tail which is said to be a venomous stinger

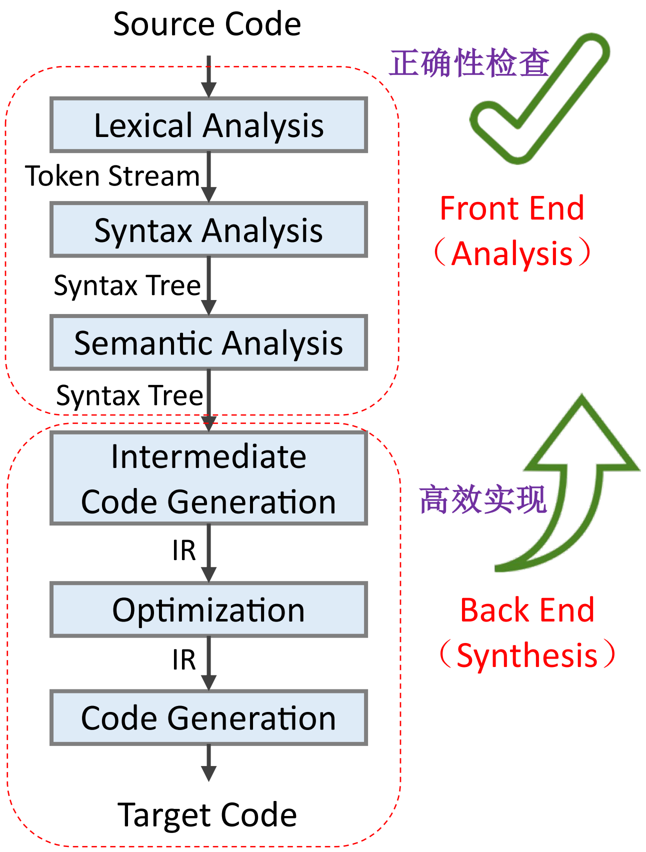


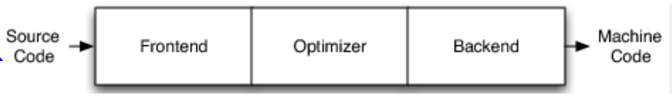


https://llvm.org/Logo.html

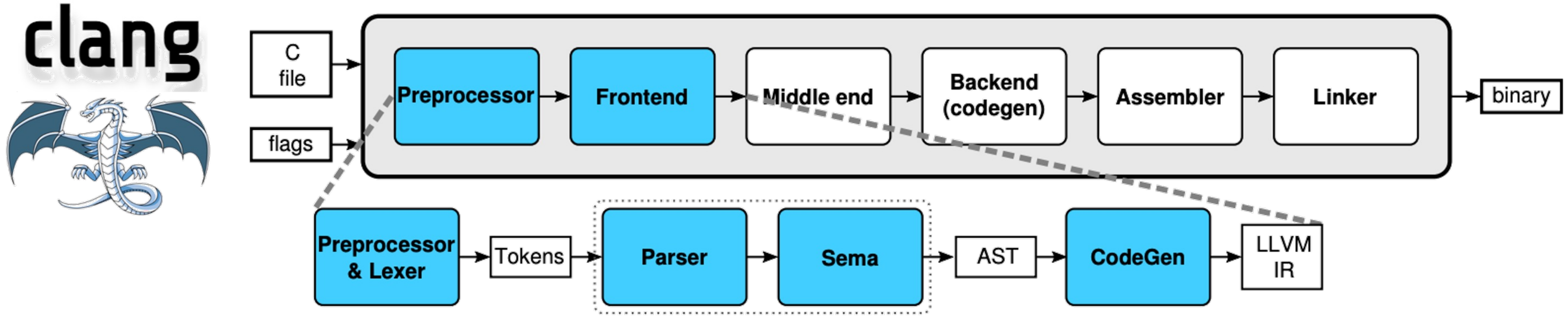
Compilation Procedure[编译过程]

- **前端**（分析）：对源程序，识别语法结构信息，理解语义信息，反馈出错信息
  - 词法分析（Lexical Analysis）词
  - 语法分析（Syntax Analysis） 语句
  - 语义分析（Semantic Analysis） 上下文
- **后端**（综合）：综合分析结果，生成语义上等价于源程序的目标程序
  - 中间代码生成（Intermediate Code Generation）
    - Intermediate representation (IR) 转换
  - 代码优化（Code Optimization） 更好
  - 目标代码生成（Code Generation）可执行





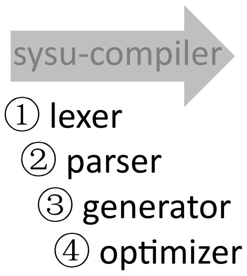
Lab[实验：编译器构造]



SYsU-lang

000_main.sysu.c

```c
int main(){
    return 3;
}
```



① lexer
② parser
③ generator
④ optimizer

a.S

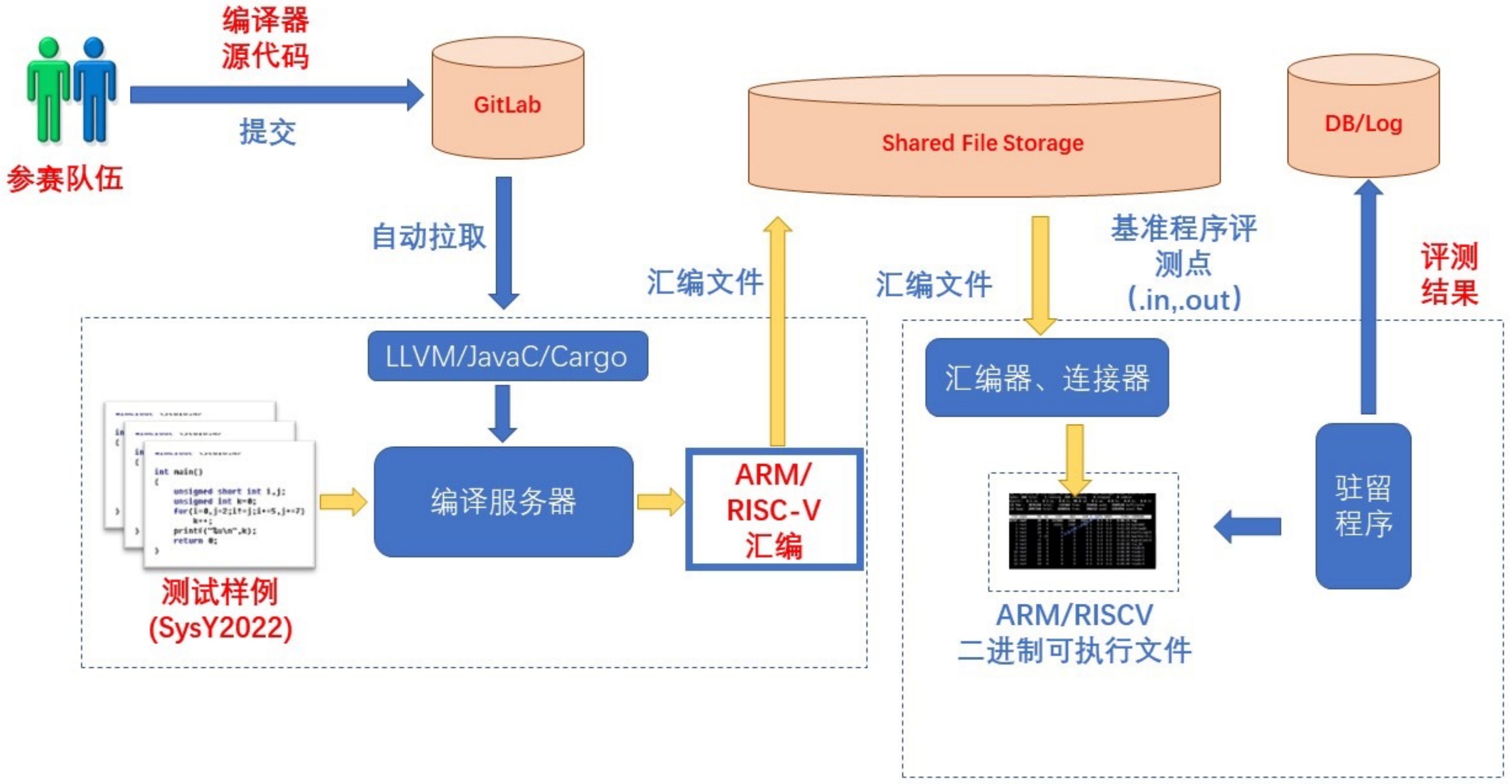
```
        .text
        .file   "-"
        .globl  main                            // -- Begin function main
        .p2align        2
        .type   main,@function
main:                                   // @main
        .cfi_startproc
// %bb.0:                               // %entry
        mov     w0, #3
        ret
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
                                        // -- End function
        .section        ".note.GNU-stack","",@progbits
```

executable

https://github.com/arcsysu/SYsU-lang2

编译系统设计赛

- 综合运用各种知识，构思并实现一个综合性编译系统

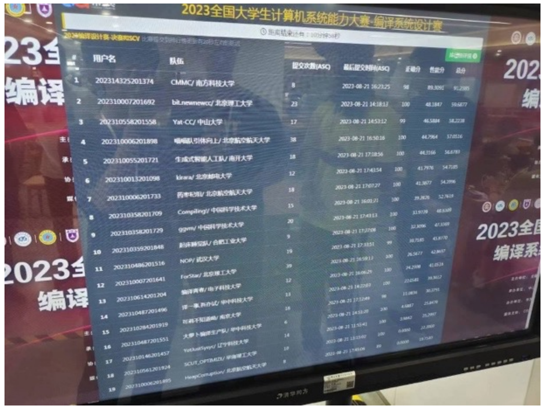


```
int main()
{
    unsigned short int i,j;
    unsigned int k=0;
    for(i=0,j=2;i!=j;i+=5,j+=7)
        k++;
    printf("%u\n",k);
    return 0;
}
```

https://compiler.educg.net/#/index

编译系统设计赛 (cont.)

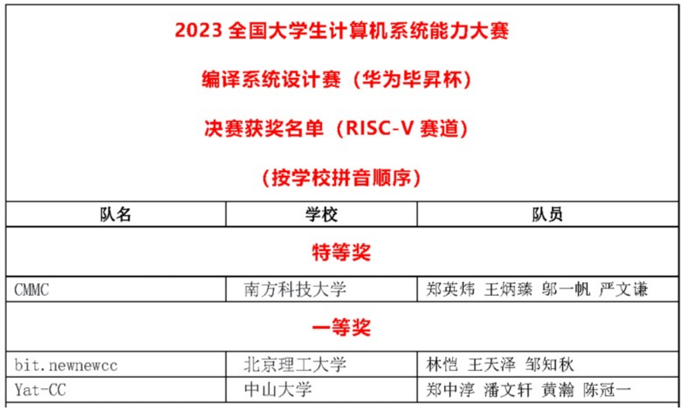
- 2023/全国一等奖，2022/优胜奖



2023全国大学生计算机系统能力大赛-编译系统设计赛

2023编译设计赛-决赛RISCV

#	用户名	队伍	提交次数(ASC)	最后提交时间(ASC)	正确分	性能分	总分
1	202314325201374	CMMC/ 南方科技大学	8	2023-08-21 16:23:25	98	89.3091	91.2385
2	202310007201692	bit.newnewcc/ 北京理工大学	23	2023-08-21 14:18:13	100	48.1847	59.6877
3	202310558201558	Yat-CC/ 中山大学	17	2023-08-21 14:53:12	99	46.5884	58.2238
4	202310006201898	喵喵队引体向上/ 北京航空航天大学	38	2023-08-21 16:50:16	100	44.7964	57.0516
5	202310055201721	生成式智能人工队/ 南开大学	18	2023-08-21 17:18:56	100	44.3166	56.6783
6	202310013201098	kirara/ 北京邮电大学	12	2023-08-21 17:43:54	100	41.7976	54.7185
7	202310006201733	[illegible]/ 北京航空航天大学	18	2023-08-21 17:07:27	100	41.3677	54.2996
8	202310358201709	Compiling/ 中国科学技术大学	15	2023-08-21 16:01:21	100	39.2826	52.7619
9	202310358201729	ggvm/ 中国科学技术大学	20	2023-08-21 17:43:13	100	33.9728	48.6309
10	202310359201848	[illegible]/ 合肥工业大学	9	2023-08-21 17:27:08	100	32.3096	47.3369
11	202310486201516	NOP/ 武汉大学	19	2023-08-21 17:18:51	99	30.7165	45.8770
12	202310007201641	ForStar/ 北京理工大学	39	2023-08-21 16:59:13	100	26.5677	42.8697
13	202310614201204	编译青春/ 电子科技大学	12	2023-08-21 16:06:29	100	24.2398	41.0516
14	202310487201496	[illegible]/ 华中科技大学	13	2023-08-21 14:22:03	100	22.0581	39.3612
15	202310284201919	[illegible]/ 南京大学	16	2023-08-21 17:12:49	98	11.0836	30.3791
16	202310487201551	大萝卜编译生产队/ 华中科技大学	6	2023-08-21 14:53:20	100	4.6887	25.8470
17	202310146201457	YetJustSysy/ 辽宁科技大学	14	2023-08-21 11:15:41	100	3.9842	25.2997
18	202310561201924	SCUT_OPTIMIZE/ 华南理工大学	10	2023-08-21 13:15:02	100	0.0000	22.0000
19	202310006201895	HeapCorruption/ 北京航空航天大学	8	2023-08-21 17:45:06	99	0.0000	19.7540



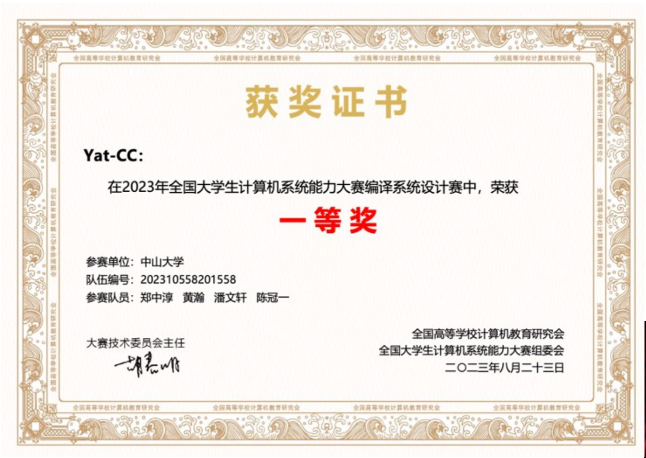
2023 全国大学生计算机系统能力大赛

编译系统设计赛（华为毕昇杯）

决赛获奖名单（RISC-V 赛道）

（按学校拼音顺序）

队名	学校	队员
特等奖		
CMMC	南方科技大学	郑英炜 王炳臻 郭一帆 严文谦
一等奖		
bit.newnewcc	北京理工大学	林恺 王天泽 邹知秋
Yat-CC	中山大学	郑中淳 潘文轩 黄瀚 陈冠一



获奖证书

Yat-CC:

在2023年全国大学生计算机系统能力大赛编译系统设计赛中，荣获

一等奖

参赛单位：中山大学

队伍编号：202310558201558

参赛队员：郑中淳 黄瀚 潘文轩 陈冠一

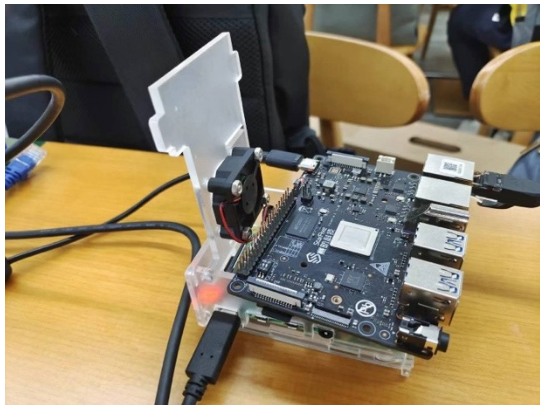
大赛技术委员会主任

[illegible]

全国高等学校计算机教育研究会

全国大学生计算机系统能力大赛组委会

二〇二三年八月二十三日

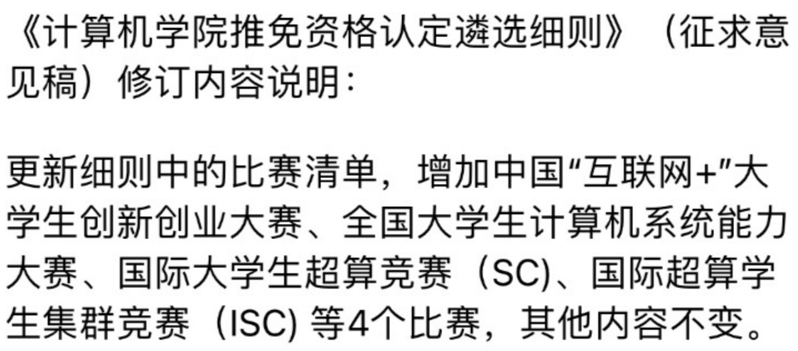


2023全国大学生计算机系统能力大赛总决赛颁奖典礼
编译系统设计赛（华为毕昇杯）
一等奖
ARM
RISC-V

https://compiler.educg.net/#/index

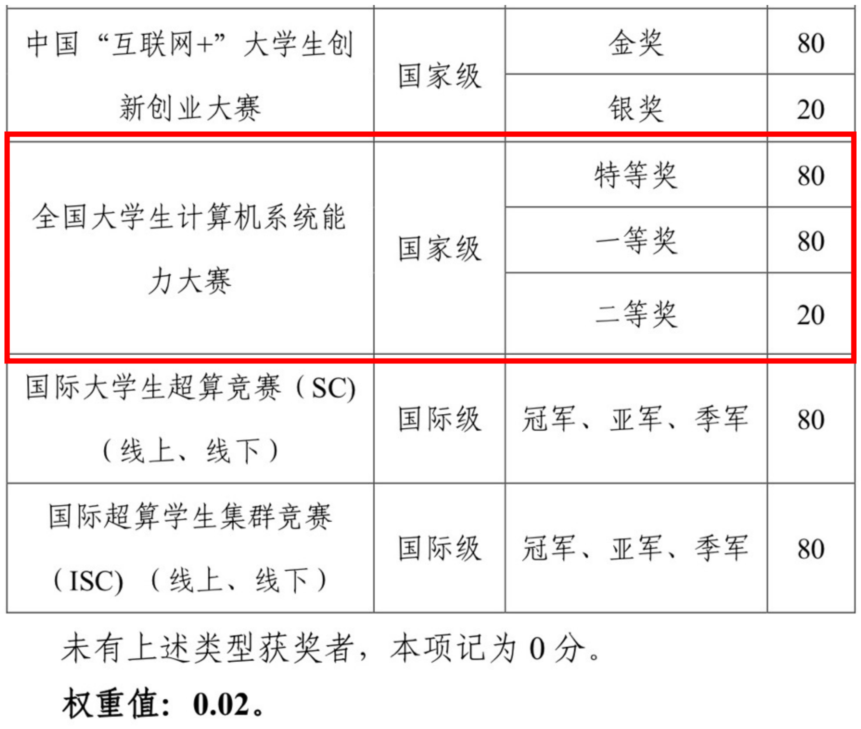
编译系统设计赛 (cont.)

- 2024开始，推免资格加分【待落实】



《计算机学院推免资格认定遴选细则》（征求意见稿）修订内容说明：

更新细则中的比赛清单，增加中国“互联网+”大学生创新创业大赛、全国大学生计算机系统能力大赛、国际大学生超算竞赛（SC)、国际超算学生集群竞赛（ISC) 等4个比赛，其他内容不变。



中国“互联网+”大学生创新创业大赛	国家级	金奖	80
		银奖	20
全国大学生计算机系统能力大赛	国家级	特等奖	80
		一等奖	80
		二等奖	20
国际大学生超算竞赛（SC)（线上、线下）	国际级	冠军、亚军、季军	80
国际超算学生集群竞赛（ISC)（线上、线下）	国际级	冠军、亚军、季军	80

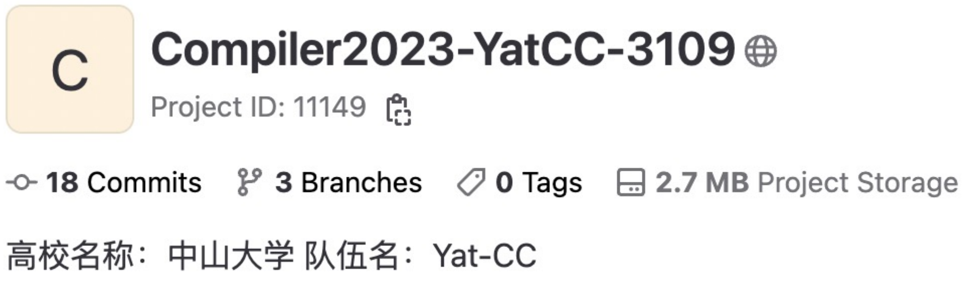
未有上述类型获奖者，本项记为 0 分。

权重值：0.02。

2024年全国大学生计算机系统能力大赛-编译系统设计赛

敬请期待

参赛队伍数量：0



Compiler2023-YatCC-3109

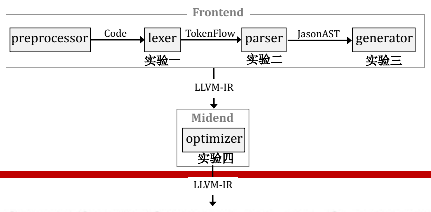
Project ID: 11149

18 Commits 3 Branches 0 Tags 2.7 MB Project Storage

高校名称：中山大学 队伍名：Yat-CC

https://compiler.educg.net/#/index

Your Well Being



Carnegie Mellon is known for its stressful environment, and we realize that the pace and expectations of 213/513 can contribute to that stress. If you find yourself having trouble keeping up, please realize the following:

- **It's Only a Class.** Your life and personal welfare are more important than your performance in this or any course.
- **Manage your Time Wisely.** Students struggling in 213/513 often follow a pattern where they fall behind and then try to catch up with a marathon effort just before an assignment is due. Instead, they start having health problems, skip or fall asleep in lectures, do poorly in this and other classes, and fall further behind. The key is to never fall behind in the first place. When an assignment goes out that is due on 2 weeks, that's because we expect it to require 2 weeks of concentrated effort to complete.
- **Take Care of Yourself.** Do your best to maintain a healthy lifestyle this semester by eating well, exercising, avoiding drugs and alcohol, getting enough sleep and taking some time to relax. This will help you achieve your goals and cope with stress.
- **Don't Resort to Cheating.** As a deadline draws near and you aren't making progress, it can become very tempting to start searching the Web or asking your friends for help. **Don't do it!** If you get caught, the consquences will be much worse than not doing the assignment at all. If you don't get caught, you will still do permanent damage to your own sense of personal integrity, your own learning, and the ability of others to put their trust in you.
- **It's OK to Ask for Help.** Some students believe that asking for help makes them look bad in the eyes of the instructor, or that it demonstrates they shouldn't be in the course in the first place. We want you to succeed, and we want to help! If you've thought about an issue and are stuck, spending a few minutes with one of the teaching staff may save you hours of frustration.
- **You are Not Alone.** All of us benefit from support during times of struggle. There are many helpful resources available on campus and in Pittsburgh. An important part of the college experience is learning how to ask for help. Asking for support sooner rather than later is often helpful.

CMU15213

Cheating/Plagiarism[学术不端]

- Unauthorized use of information
  - Borrowing code: by copying, retyping, looking at a file
  - Describing: verbal desc. of code from one person to another
  - Searching the Web for solutions, discussions, tutorials, blogs …
  - Reusing your code from a previous semester …
  - …
- Unauthorized supplying of information
  - Providing copy: Giving a copy of a file to someone
  - Providing access …
  - …
- Collaborations beyond high-level, strategic advice
  - Anything more than block diagram or a few words
  - …

CMU15213

Why a Big Deal?

- This material is best learned by doing
  - Even though that can, at times, be difficult and frustrating
  - Starting with a copy of a program and then tweaking it is very different from writing from scratch
    - Planning, designing, organizing a program are important skills
- We are the gateway to other system courses
  - Want to make sure everyone completing the course has mastered the material
- Industry appreciates the value of this course
  - We want to make sure anyone claiming to have taken the course is prepared for the real world
- Working in teams and collaboration is an important skill
  - But only if team members have solid foundations
  - This course is about foundations, not teamwork

Cheating: Consequences[后果]

- Penalty for cheating:
  - Best case: -100% for assignment
    - ***You would be better off to turn in nothing***
  - Worst case: Removal from course with failing grade
    - This is the default
  - University-level involvement (from notification to serious things)
  - Loss of respect by you, the instructors and your colleagues
  - ***If you do cheat – come clean asap!***

- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism
  - In Fall 2015, 20 students were caught cheating and failed the course.
    - Some were **expelled** from the University
  - In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.
  - In May 2019, we gave an AIV to a student who took the course in Fall 2018 for unauthorized coaching of a Spring 2019 student. His grade was changed retroactively.

- Don't do it!
  - Manage your time carefully
  - Ask the staff for help when you get stuck
  - We will help you! We will give you extensions! We want you to succeed.

CMU15213

Compilation Principle
编 译 原 理

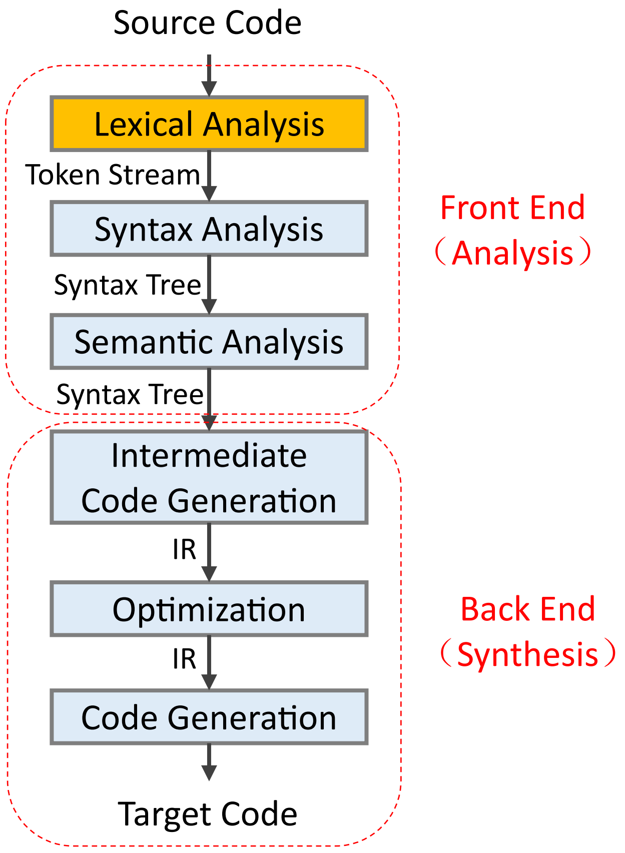
第1讲：词法分析(1)

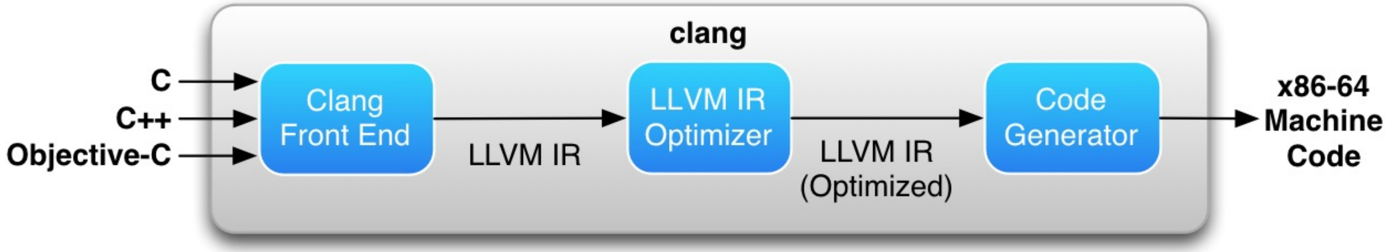
张献伟

xianweiz.github.io

DCS290, 2/29/2024

Structure of a Typical Compiler[结构]





Example

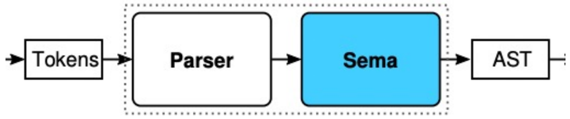
```c
void main(){
  int a, b, c;
  if (b == c)
    return 1;
}
```

$clang -cc1 -dump-tokens test.c

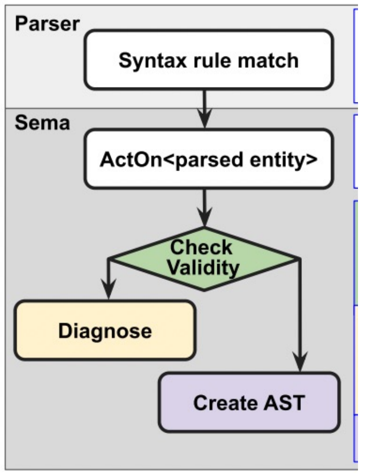
```
void 'void'      [StartOfLine]  Loc=<parse.c:1:1>
identifier 'main'        [LeadingSpace] Loc=<parse.c:1:6>
l_paren '('             Loc=<parse.c:1:10>
r_paren ')'             Loc=<parse.c:1:11>
l_brace '{'             Loc=<parse.c:1:12>
int 'int'        [StartOfLine] [LeadingSpace]   Loc=<parse.c:2:3>
identifier 'a'   [LeadingSpace] Loc=<parse.c:2:7>
comma ','               Loc=<parse.c:2:8>
identifier 'b'   [LeadingSpace] Loc=<parse.c:2:10>
comma ','               Loc=<parse.c:2:11>
identifier 'c'   [LeadingSpace] Loc=<parse.c:2:13>
semi ';'                Loc=<parse.c:2:14>
if 'if'  [StartOfLine] [LeadingSpace]   Loc=<parse.c:3:3>
l_paren '('      [LeadingSpace] Loc=<parse.c:3:6>
identifier 'b'          Loc=<parse.c:3:7>
equalequal '=='  [LeadingSpace] Loc=<parse.c:3:9>
identifier 'c'   [LeadingSpace] Loc=<parse.c:3:12>
r_paren ')'             Loc=<parse.c:3:13>
return 'return'  [StartOfLine] [LeadingSpace]   Loc=<parse.c:4:5>
numeric_constant '1'     [LeadingSpace] Loc=<parse.c:4:12>
semi ';'                Loc=<parse.c:4:13>
r_brace '}'      [StartOfLine]  Loc=<parse.c:5:1>
eof ''          Loc=<parse.c:5:2>
```

$clang -Xclang -ast-dump -fsyntax-only test.c

```
`-FunctionDecl 0x27999470 <parse.c:1:1, line:5:1> line:1:6 main 'void ()'
  `-CompoundStmt 0x27999800 <col:12, line:5:1>
    |-DeclStmt 0x279996f8 <line:2:3, col:14>
    | |-VarDecl 0x27999570 <col:3, col:7> col:7 a 'int'
    | |-VarDecl 0x279995f0 <col:3, col:10> col:10 used b 'int'
    | `-VarDecl 0x27999670 <col:3, col:13> col:13 used c 'int'
    `-IfStmt 0x279997e8 <line:3:3, line:4:12>
      |-BinaryOperator 0x27999780 <line:3:7, col:12> 'int' '=='
      | |-ImplicitCastExpr 0x27999750 <col:7> 'int' <LValueToRValue>
      | | `-DeclRefExpr 0x27999710 <col:7> 'int' lvalue Var 0x279995f0 'b' 'int'
      | `-ImplicitCastExpr 0x27999768 <col:12> 'int' <LValueToRValue>
      |   `-DeclRefExpr 0x27999730 <col:12> 'int' lvalue Var 0x27999670 'c' 'int'
      `-ReturnStmt 0x279997d8 <line:4:5, col:12>
        `-ImplicitCastExpr 0x279997c0 <col:12> 'void' <ToVoid>
          `-IntegerLiteral 0x279997a0 <col:12> 'int' 1
```



Sema is tight coupling with parser



https://llvm.org/devmtg/2019-10/slides/ClangTutorial-Stulova-vanHaastregt.pdf

What is Lexical Analysis[词法分析]？

- Example:

```
/* simple example */
if (i == j)
  z = 0;
else
  z = 1;
```

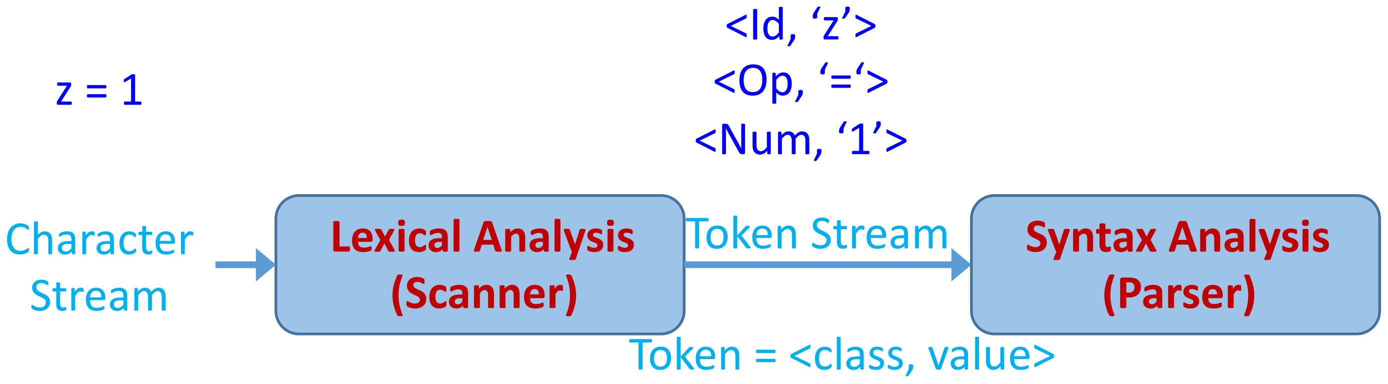
- Input[输入]: a string of characters
  - “*if* (*i* == *j*)\n\t*z* = 0; \n*else*\n\t*z* = 1; \n”
- Goal[目标]: partition the string into a set of substrings
  - Those substrings are **tokens**
- Steps[步骤]
  - Remove comments: ~~/* simple example */~~
  - Identify substrings: ‘if’ ‘(‘ ‘i’ ‘==‘ ‘j’ ……
  - Identify **token classes**: (keyword, ‘if’), (LPAR, ‘(‘), (id, ‘i’) ……

What is a token[词]?

- **Token**: a "word" in language (<u>smallest unit with meaning</u>)
  - Categorized into classes according to its role in language
  - Token classes in English[自然语言]
    - Noun, verb, adjective, …
  - Token classes in a programming language[编程语言]
    - Number, keyword, whitespace, identifier, …
- Each **token class** corresponds to a set of strings[类: 集合]
  - Number: a non-empty string of digits
  - Keyword: a fixed set of reserved words ("for", "if", "else", …)
  - Whitespace: a non-empty sequence of blanks, tabs, newlines
  - Identifier: user-defined name of an entity to identify
    - Q: what are the rules in C language?

Lexical Analysis: Tokenization[分词]

- Lexical analysis is also called **Tokenization** (also called Scanner)[扫描器]
  - Partition input string into a sequence of tokens
  - Classify each token according to its role (token class)
    - **Lexeme**[词素]: an instance of the token class, e.g. ‘z’, ‘=’, ‘1’
- Pass tokens to syntax analyzer (also called Parser)[分析器]
  - Parser relies on token classes to identify roles (e.g., a *keyword* is treated differently from an *identifier*)



Lexical Analyzer: Design[设计]

- Define a finite set of token classes[定义token类别]
  - Describe all items of interest
  - Depends on language, design of parser
  - "*if* (*i* == *j*)\n\tz = 0; \nelse\n\tz = 1; \n"
    - Keyword, identifier, whitespace, integer

- Label which string belongs to which token class[识别]

```
if (i == j)
  z = 0;
else
  z = 1;
```

'==' or '='?

keyword or identifier?

Lexical Analyzer: Implementation[实现]

- An implementation must do two things
  - Recognize the token class the substring belongs to[识别分类]
  - Return the value or lexeme of the token[返回对应值]
- A token is a tuple (class, lexeme)[二元组]

- The lexer usually discards "non-interesting" tokens that don't contribute to parsing[丢弃无意义词]
  - e.g., whitespace, comments
- If token classes are non-ambiguous, tokens can be recognized in a single left-to-right scan of input string
- Problem can occur when classes are ambiguous[歧义]

Ambiguous Tokens in C++

- C++ template syntax
  - Foo<Bar>
- C++ stream syntax
  - cin >> var

Template: a blueprint or formula for creating a generic class or a function.
Templates are expanded at compiler time, similar to macros.

```
Template <typename T>
T getMax(T x, T y) {
   return (x > y) ? x : y;
}

int main (int argc, char* argv[]) {
    getMax<int>(3, 7);
    getMax<double>(3.0, 2.0);
    getMax<char>('g', 'e');

    return 0;
}
```

- Ambiguity
  - Foo<Bar<Bar>>
  - cin >> var
  - Q: Is '>>' a stream operator or two consecutive brackets?

中山大學 SUN YAT-SEN UNIVERSITY

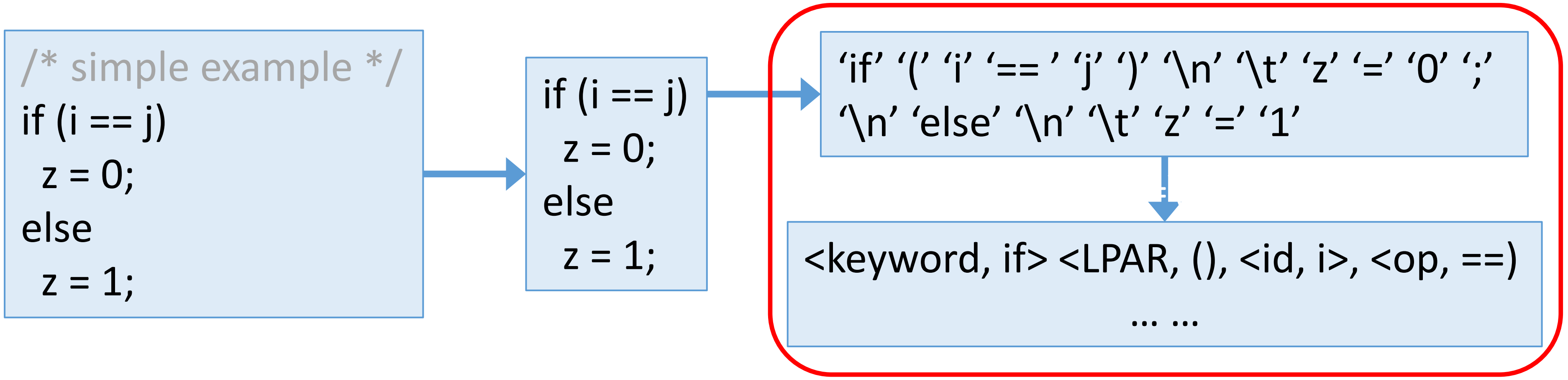
NSCC GZ

Look Ahead[展望]

- “look ahead” may be required to resolve ambiguity[展望消除歧义]
  - Extracting some tokens requires looking at the larger context or structure[需要上下文或语法结构]
  - Structure emerges only at parsing stage with parse tree[后一阶段才有]
  - Hence, sometimes feedback from parser needed for lexing
    - This complicates the design of lexical analysis
    - Should minimize the amount of look ahead
- Usually tokens do not overlap[通常无重叠]
  - Tokenizing can be done in one pass w/o parser feedback
  - Clean division between lexical and syntax analyses

Summary: Lexer

- Lexical analysis
  - Partition the input string to lexeme
  - Identify the token class of each lexeme
- Left-to-right scan => look ahead may be required
  - In reality, lookahead is always needed
  - The amount of lookahead should be minimized



```
/* simple example */
if (i == j)
  z = 0;
else
  z = 1;
```

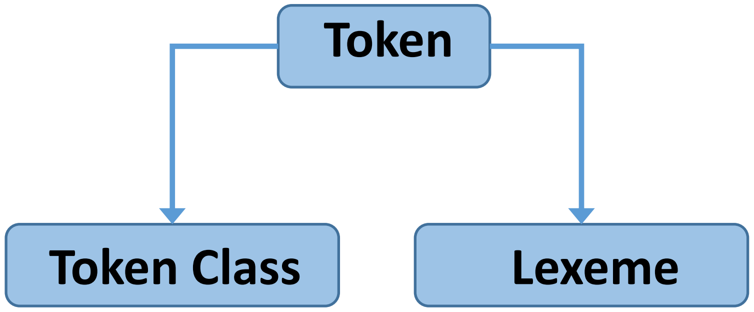
```
if (i == j)
  z = 0;
else
  z = 1;
```

```
'if' '(' 'i' '== ' 'j' ')' '\n' '\t' 'z' '=' '0' ';'
'\n' 'else' '\n' '\t' 'z' '=' '1'
```

```
<keyword, if> <LPAR, (), <id, i>, <op, ==)
… …
```

Token Specification[定义]

- Recognizing token class: how to describe string patterns
  - i.e., which set of strings belong to which token class?
  - Use regular expressions[正则表达式] to define token class
- **Regular Expression** is a good way to specify tokens
  - Simple yet powerful (able to express patterns)
  - Tokenizer implementation can be generated automatically from specification (using a translation tool)
  - Resulting implementation is provably efficient



String patterns describing the class

Language: Definition

- **Alphabet** ∑[字母表]: a finite set of symbols
  - Symbol: *letter*, *digit*, *punctuation*, …
  - Example: {0, 1}, {a, b, c}, ASCII
- **String**[串]: a finite sequence of symbols drawn from ∑
  - i.e., sentence or word
  - Example: *aab* (length = 3), *ε* (empty string, length = 0)
- **Language**[语言]: a set of strings of the characters drawn from ∑
  - ∑ = {0, 1}, then {}, {01, 10}, {1, 11, 1111, …} are all languages over ∑
  - {ε} is a language
  - Φ, empty set is also a language

Language: Example

- Examples:
  - Alphabet ∑ = (set of) English characters
    - Language *L* = (set of) English sentences
  - Alphabet ∑ = (set of) Digits, +, -
    - Language *L* = (set of) Integer numbers

- Languages are <u>subsets of all</u> possible strings
  - Not all strings of English characters are (valid) sentences
    - aaa bbb ccc
  - Not all sequences of digits and signs are integers
    - 125+, 1-25

Language: Operations[语言运算]

- In lexical analysis, the most important operations on languages are <u>union</u>, <u>concatenation</u> and <u>closure</u>
- **Union**[并]: similar operation on sets
- **Concatenation**[连接]: all strings formed by taking a string from the first language and a string from the second language in <u>all possible ways</u>, and concatenating them
- **Kleene closure**[闭包]: L^*, where *L* is the language, is the set of strings you get by concatenating *L* <u>zero or more</u> times
  - $L^0 = \{\varepsilon\}$, $L^i = L^{i-1}L$
  - L^+: the same as Kleene closure, but without L^0
    - $L \cup L^2 \cup L^3 \cup \ldots$
    - ε won't be in L^+ unless it is in *L* itself

Example

- $\sum_1 = \{0, 1\}$, $\sum_2 = \{a, b\}$ ==> $L_1 = \{0, 1\}$, $L_2 = \{a, b\}$
- $L_1 \cup L_2$

  $= \{0, 1\} \cup \{a, b\} = \{0, 1, a, b\}$
- $L_1 L_2$

  $= \{0, 1\}\{a, b\} = \{0a, 0b, 1a, 1b\}$
- L_1^3

  $= \{0, 1\}^3 = \{0, 1\}\{0, 1\}\{0, 1\} = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- L_1^*

  $= L_1^0 \cup L_1^1 \cup L_1^2 \cup L_1^3 \cup \ldots = \{\varepsilon\} \cup \{0, 1\} \cup \{0, 1\}^2 \cup \{0, 1\}^3$

  $= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots\}$
- L_1^+
  - $= L_1^* - L_1^0 = \{0, 1\} \cup \{0, 1\}^2 \cup \{0, 1\}^3$
  - $= \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots\}$

Language: Example (cont.)

- L = {A, B, …, Z, a, b, …, z}, D = {0, 1, …, 9}
  - L and D are also languages whose strings happen to be of length one
  - Some other languages that can be constructed from L and D are
- $L \cup D$: the set of letters and digits, i.e., language with 62 strings of length one
- LD: the set of 520 strings of length two, each is one letter followed by one digit
- L^4: the set of all 4-letter strings
- L^*: the set of all strings of letters, including ε, the empty string
- $L(L \cup D)^*$: the set of all strings of letters and digits beginning with a letter
- D^+: the set of all strings of one or more digits

Identifiers can be described by giving names to sets of letters and digits and using the language operators

Regular Expressions & Languages[正则]

- **Regular expressions** are to describe all the languages that can be built from the operators applied to the symbols of some alphabet
- Regular Expression is a <u>simple</u> notation
  - Can express simple patterns (e.g., repeating sequences)
  - Not powerful enough to express English (or even C)
  - But powerful enough to express tokens (e.g., identifiers)
- Languages that can be expressed using regular expressions are called **Regular Languages**
- More complex languages need more complex notations
  - More complex languages and expressions[非正则] will be covered later

Atomic REs[原子表达式]

- Atomic
  - Smallest RE that cannot be broken down further

- **Epsilon or ε** character denotes a zero length string
  - ε = {“”}
- **Single character** denotes a set of one string
  - ‘c’ = {“c”}
- Empty set is { } = ϕ, not the same as ε
  - Size(ϕ) = 0
  - Size(ε) = 1
  - Length(ε) = 0

Compound REs[组合表达式]

- **Compound**
  - Large REs built from smaller ones
- Suppose *r* and *s* are REs denoting languages L(*r*) and L(*s*)
  - (*r*) is a RE denoting the language L(*r*)
    - We can add additional () around expressions without changing the language they denote
  - (*r*)|(*s*) is a RE denoting the language L(*r*) ∪ L(*s*)
  - (*r*)(*s*) is a RE denoting the language L(*r*)L(*s*)
  - $(r)^*$ is a RE denoting the language $(L(r))^*$
- REs often contain unnecessary (), which could be dropped
  - **(A) ≡ A**: A is a RE
  - $(a)|((b)^*(c)) \equiv a|b^*c$