讲稿（教学内容、步骤）

第 1 章 引论

1. 编译原理课程：介绍“高级语言程序”到“低级语言程序”转换的方法及原理。

2. 课程地位及作用：
   (1) 是计算机软件技术的基础。
   (2) 相关技术还大量用于人工智能、多媒体技术、数据库等领域。
   (3) 有助于程序员深入思考和理解程序的底层机制，多维度地合理设计问题的解决方案（语法维度、代码优化维度、与硬件结合的维度等），工作面试、研究生复试的必考专业知识点，是专业能力的体现。

3. 课程目标

课程目标	毕业要求指标点
目标1：掌握高级程序语言到低级程序语言变换的基本概念和编译过程；掌握编译过程中词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成的实现原理和方法。	1.3 系统掌握计算机领域的基础理论及专业知识，包括计算机硬件、软件及系统等方面内容，具备理解计算机复杂工程问题的能力，能够运用所学知识进行计算机问题求解。
目标2：掌握编译程序设计的基本过程与实现方法，能够针对特定需求完成编译程序或相关模块的设计与实现。	3.2 能够合理地组织数据、有效地存储和处理数据，正确地设计算法以及对算法的分析和评价。
目标3：学习编译程序构造的算法，能够对不同方法的优劣进行对比和分析，形成具体的解决方案，并独立撰写实验报告。	10.1 能够就计算机工程领域复杂工程问题独立撰写实验报告、研究报告、说明书、项目计划书、学术论文等。

4. 编译程序（又称“编译器”）：语言的翻译器
   高级语言的源程序 --> 低级语言的目标程序

```
for (int i = 0; i < producNum; i·
    string c = grammar[i].substr
    if (c[0] != '@')
    {
        cout << "FIRST(" << gram
        for (char fi : producFir
            cout << " " << fi;
    }
```

```
嗓□1襴8s□敷H婦$8f畤□槃?  恣蓑$8   H?%□
  ??  H堉鏖2  H咿H堪$8□剁?  ?□□匪?  閍?
剄□  峪裙?□嘘  <o□膀  <t□敷t□<iu
€y  l□勴□  H堉椓□  H咿H壟□?  ? 凐1□?□  凐:
 H堉?H  l姓l嘲?  H堉??  H壟H盯璷T□??  1
€:l□刪□  H璞□D墦(D塭8墦@璷PH婦$8??  ?%
 H咿H壟□劑□  H婄□€:l□刋□  璷T闡?  □禔  <
```

5. 编译程序的作用：使编程者不必考虑与机器有关的细节。

6. 程序设计语言
   高级语言
   - 与自然语言比较接近的语言
     - ✓ 过程式语言：C, Pascal, Fortran, ADA
     - ✓ 对象式语言：Java, C++, python 等
     - ✓ 函数式语言：LISP
     - ✓ 逻辑式语言：Prolog
   - 特点：执行效率低、编制效率高

低级语言

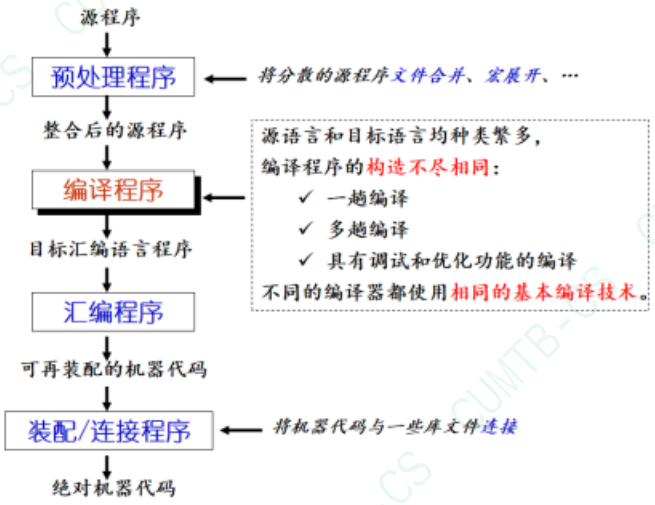
- 特定的计算机系统所固有的语言：机器语言、汇编语言
- 特点：执行效率高、编制效率低

7. 编译技术的应用场景
   (1) 制作编译器（比尔盖茨早期最主要的成就是写了一个 Basic 的解释器）。
   (2) 苹果、谷歌、微软等技术巨头的核心能力，都是拥有自己的语言和生态，编译技术是其重要支撑。
   (3) Office、报表软件、工资管理等应用程序开发中遇到用户自定义公式来判断工作流方向时，需要使用编译技术。
   (4) 基础设施类软件（如：数据库软件、大数据平台、ETL 软件）开发中，需要编译技术提供软件自有的语言功能（如 SQL）。
   (5) Java Hibernate 的 HQL 解析（抽象语法树生成和语义分析），需要编译技术。
   (6) Java Spring 对注解的支持和字节码的动态生成，需要编译技术。
   (7) PHP 使用模板引擎实现界面设计与代码分离，模板引擎对模板进行编译形成 PHP 代码。了解编译技术，你可以写出更符合领域需求的模板引擎。
   (8) 对用户输入的解析及有效性判别，需要编译技术。
   (9) 防止代码注入攻击，使用正则表达式过滤传入的参数，需要编译技术。
   (10) 两种高级语言之间的翻译，需要编译技术。
   (11) 分析日志文件，需要编译技术。

8. 编译系统的发展契机
   (1) 人工智能的再次兴起（2021 年是人工智能的普及之年）
   人工智能加速芯片+人工智能算法开发，对程序设计语言和编译程序提出了更高要求。
   (2) 国产芯片五年计划（2020 年 8 月）
   - 2020 年新增了超过 6 万家芯片相关企业
   - 到 2025 年实现 70%的国产芯片自给

   (3) 面向应用/硬件的领域特定语言、软硬件协同的编译系统优化
   (4) 程序设计语言众多，不断出现新语言（www.tiobe.com/tiobe-index/）
   (5) 程序语言自身也不断发展，新版本具有新特征

9. 高级语言程序的处理过程



10. 编译程序的历史和发展

(1) 20 世纪 50 年代早期

- 将计算公式翻译成机器码

(2) 20 世纪 50 年代中期

- 出现了高级语言的编译程序，如：FORTRAN

(3) 20 世纪 50 年代后期

- 出现了编译程序的编译程序，如：LEX、YACC

(4) 20 世纪 60 年代

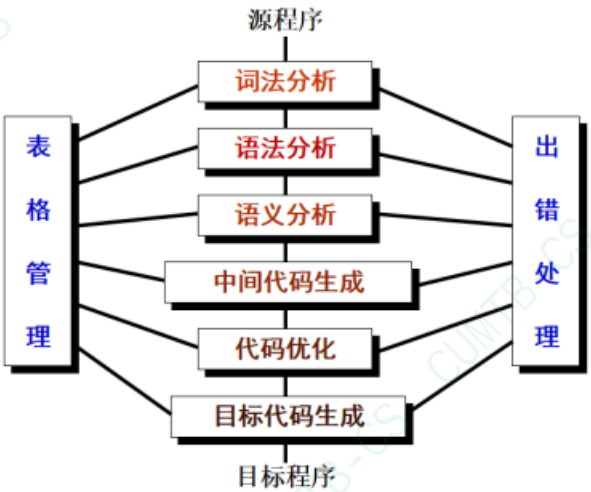
- 用自展技术构造编译程序

如：1971 年 PASCAL，用被编译语言书写其自身的编译程序

(5) 发展方向

- 并行语言的并行编译
- 自动并行编译技术（将串行程序转换成并行程序）

11. 编译过程



(1) 词法分析

任务：从左到右读入源程序的每个字符，对构成源程序的字符流进行扫描和分解，从而识别出一个个单词（Token / 单词符号 / 符号）。

单词是具有独立意义的最小语法单位。

如：标识符、保留字（关键字）、算符、界符、常数

【举例】

词法分析举例

```
float  bmi, w = 75.0, h = 1.8;
bmi = w / (h*h);
if(bmi >= 30){
    cout << "fat" ;
}
```

```
保留字  float
标识符  bmi
逗号    ,
标识符  w
赋值号  =
实数    75.0
逗号    ,
标识符  h
赋值号  =
实数    1.8
分号    ;
标识符  bmi
赋值号  =
标识符  w
除号    /
左括号  (
标识符  h
    ⋮
```

(2) 语法分析

任务： 在词法分析的基础上，将单词序列分解成各类**语法短语**。

语法短语：程序中的各**语法**成分，如“程序”、“语句”、“表达式”等。

语法：由程序语言基本符号组成程序中各个语法成分的一组规则。它是语法分析的依据。

- 一般语法规则：由单词符号构成语法成分的规则；
- 词法规则：由基本符号构成的符号书写规则。

语法规则（可以递归表示）

1）任何标识符是表达式

2）任何整数是表达式

任何实数是表达式

若表达式 1 和表达式 2 都是表达式

3）表达式 1+表达式 2 仍是表达式

4）表达式 1*表达式 2 仍是表达式

5）（表达式 1） 仍是表达式

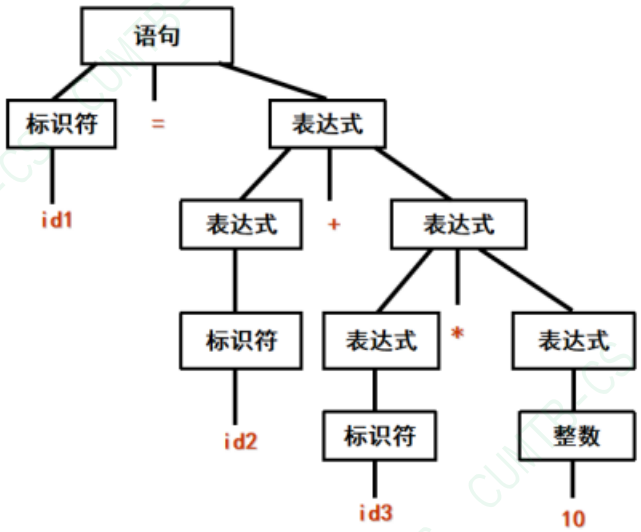
6）标识符=表达式 是语句

7）while（表达式）do 语句 是语句

8）if（表达式）then 语句 else 语句 是语句

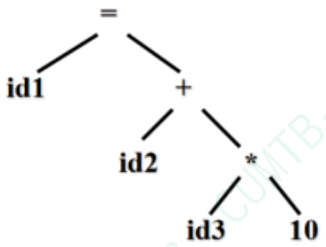
【讨论】利用这组语法规则，对语句 id1=id2+id3*10 进行语法分析

id1=id2+id3*10 的语法分析过程：试图构造出一棵完整的语法树



利用语法规则，进行语法分析（构造语法树），来确定整个输入串是否构成一个语法上正确的程序。

id1=id2+id3*10 语法树的简化形式：



(3) 语义分析

任务：审查源程序有无语义错误，为代码生成阶段收集类型信息。

主要功能：报语义错误、类型检查、类型转换等。

语义：程序设计语言中按语法规则构成的各个语法成分的意义。

- 静态语义：编译时刻即可确定的语法成分含义。
- 动态语义：运行时刻才能确定的语法成分含义。

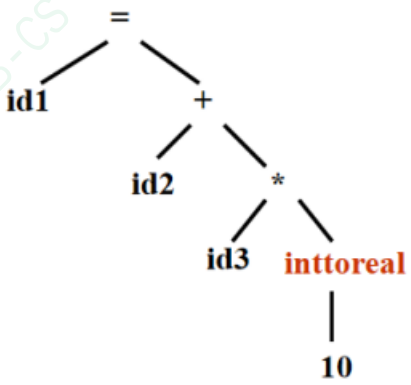
【举例】

- ✓ 我学习编译原理。(语义正确)
- ✓ 编译原理学习我。(语义错误)

【举例】类型检查和转换

```
float  first, count, sum
…

sum = first + count * 10
```



(4) 中间代码生成

任务：在语法和语义分析之后，将源程序变成一种“内部表示形式”。

中间代码：一种结构简单、含义明确的记号系统。

中间代码的特征：

1）结构简单、含义明确

2）复杂性介于源语言和目标语言之间

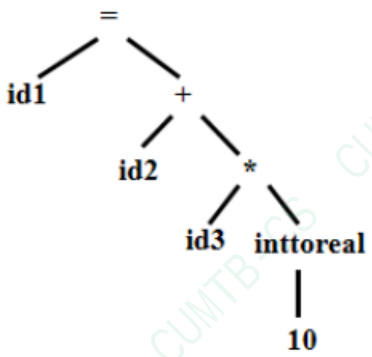
3）容易生成；

4）容易将它翻译成目标代码。

常见的中间代码：四元式

（运算符，运算对象 1，运算对象 2，结果）

【举例】



例：源程序 sum = first + count * 10

四元式序列：

(inttoreal , 10 , - , t1)
(* , id3 , t1 , t2)
(+ , id2 , t2 , t3)
(= , t3 , - , id1)

(5) 代码优化

任务：对中间代码进行等价变换，使之更为高效（时间、空间）。

【举例】

优化举例：

(inttoreal , 10 , - , t1)
(* , id3 , t1 , t2)
(+ , id2 , t2 , t3)
(:= , t3 , - , id1)

↓

(* , id3 , 10.0 , t2)
(+ , id2 , t2 , id1)

↓

(* , id3 , 10.0 , t1)
(+ , id2 , t1 , id1)

【讨论】该样例中存在哪些优化？

(6) 目标代码生成

任务：把中间代码变换成特定机器上的 绝对指令代码 或 可重定位的机器指令代码 或 汇编指令代码。

特点：

1）与硬件系统结构和指令含义有关，涉及到硬件系统功能部件的运用、机器指令选择、指令调度、变量的存储空间分配、寄存器分配等。

2）是基于语义的等价变换，不是结构上的变换。

【举例】

sum := first + count * 10

↓

中间代码：

(*, id3, 10.0, t1)
(+, id2, t1, id1)

↓

目标代码：

```
MOVF    id3, R2
MULF    #10.0, R2
MOVF    id2, R1
ADDF    R2, R1
MOV     R1, id1
```

(7) 表格管理

任务：保存源程序的各种信息。（编译的各阶段工作均需要查找、更新、构造表格。）

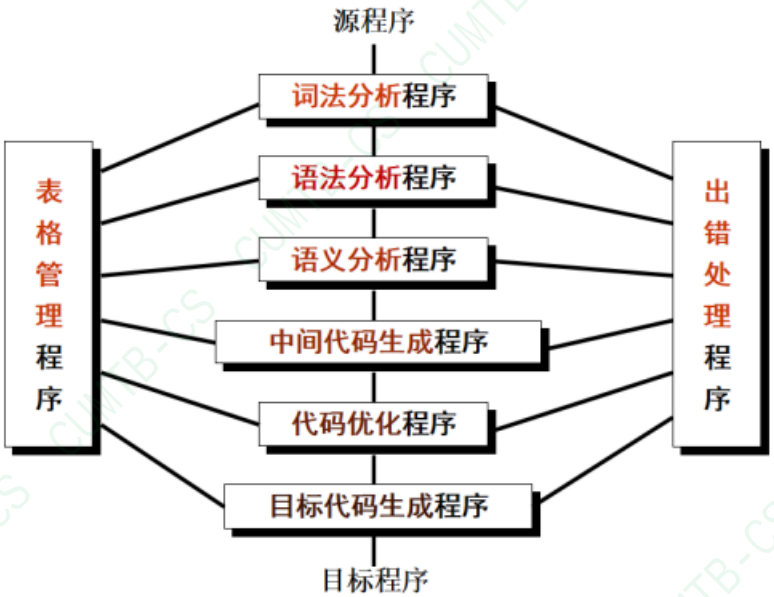
(8) 出错处理

- 报告源程序中错误的性质、地点，将错误所造成的影响限制在尽可能小的范围。
- 有些编译程序还可以自动纠错。
- 一个程序是正确的，包括两层含义：
  - ✓ 书写正确（合乎语法规则）
  - ✓ 含义正确（合乎语义规则）

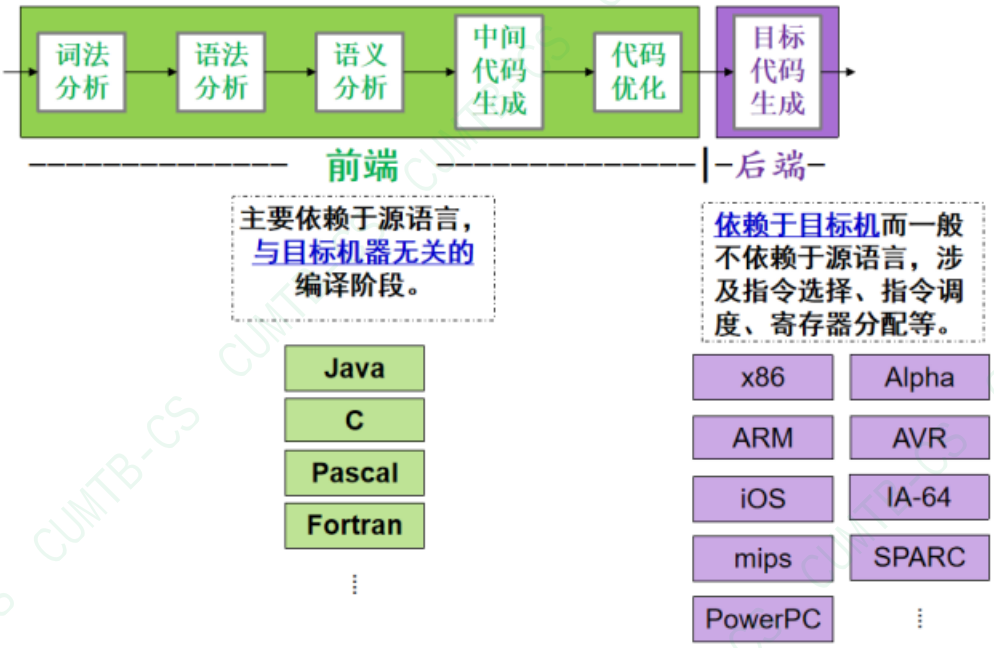
(9) 说明：大多数实用的编译程序都采用六个处理阶段。

还有些编译程序没有“中间代码生成”和“代码优化”。

12. 编译程序的结构



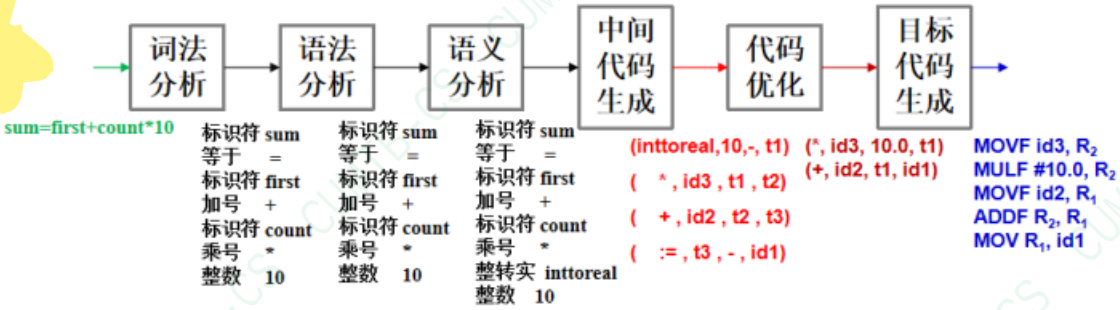
13. 编译阶段的组合



14. 遍（趟）：对源程序或其等价的中间语言程序从头到尾扫视并完成规定任务的过程。每一遍扫视可完成编译的一个阶段或多个阶段工作。

- 多遍编译：占内存少，逻辑结构清晰，耗时长
- 一遍编译：占内存多，逻辑结构不清晰，耗时短

【举例】



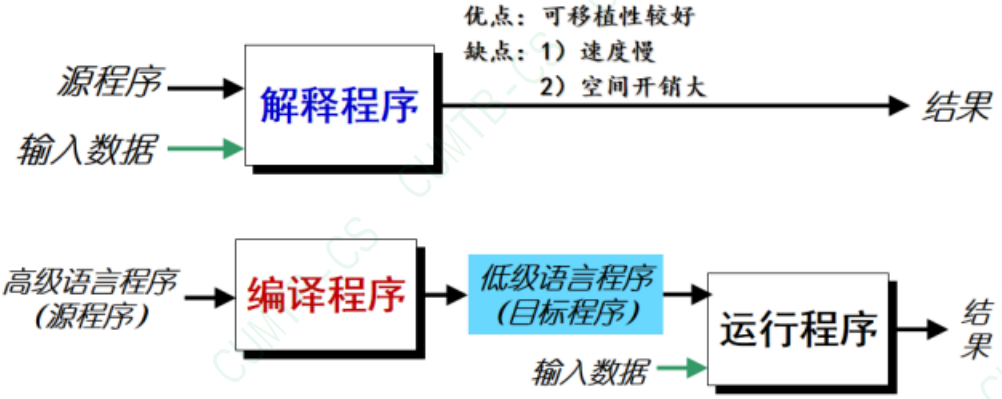
15. 解释程序

接受高级语言程序，并立即运行这个源程序。工作模式是一个个获取、分析并执行源程序语句。

【举例】BASIC 语言解释程序，LISP 解释程序，SQL 解释程序，Java 语言中的 BYTECODE 解释程序。

有些语言既有编译程序，又有解释程序。如 java。

16. 解释程序与编译程序的比较



编译与解释的根本区别：是否生成目标代码。

17. 处理源程序的软件工具【自学】

(1) 语言的结构化编辑器
- 正文编辑、修改
- 对源程序正文进行分析（检查用户输入是否正确、自动提供关键字、检查括号的匹配情况）

(2) 语言程序的调试工具
- 了解程序执行的结果与编程人员的意图是否一致
- 允许用户一行一行跟踪程序，查看变量值的变化

(3) 程序格式化工具
- 分析源程序，并使程序结构变得清晰可读（如缩排）

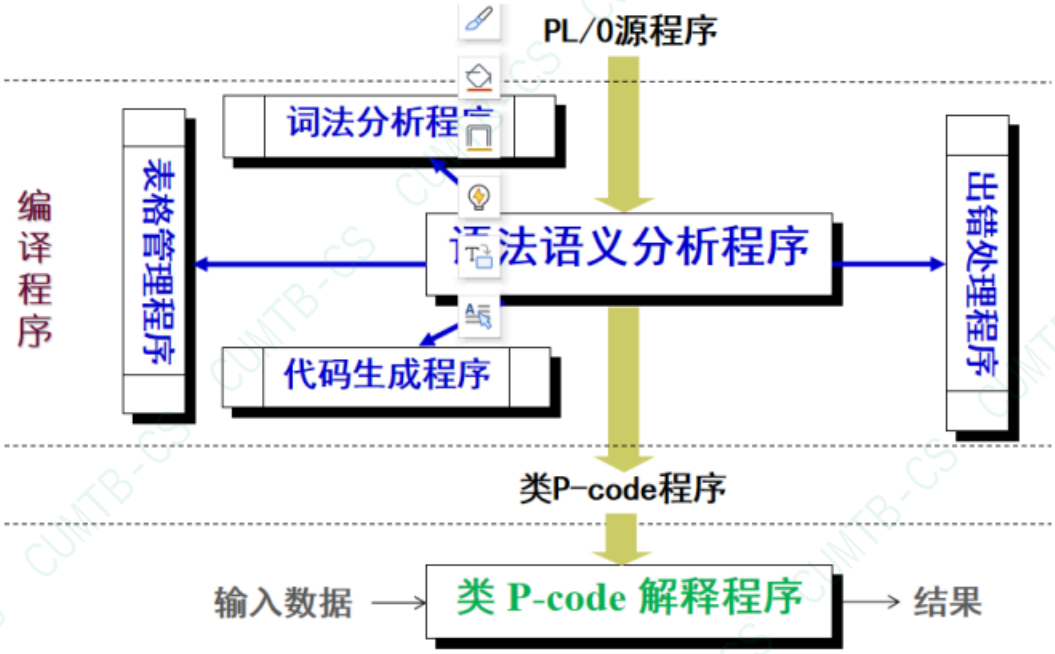
(4) 语言程序测试工具
- 静态分析器：不运行源程序，就可以发现其中潜藏的错误或异常。
- 动态分析器：对源程序进行分析，把记录和显示程序执行轨迹的语句或函数插入源程序，将运行结果与期望结果进行比较和分析。

(5) 程序理解工具
- 对程序进行分析，确定模块间调用关系，并画出控制流图。

(6) 高级语言之间的转换工具
- 将一种高级语言程序转换成另一种高级语言程序。

18. PL/0 语言编译系统的构成
- PL/0 以语法语义分析程序为核心。
- 用表格管理程序建立变量、常量和过程标识符的说明与引用之间的信息联系。
- 用出错处理程序给出源程序出错的位置和错误性质。

19. PL/0 编译系统的结构



- PL/0 以“语法语义分析程序”为核心，“词法分析程序”和“代码生成程序”都作为一个独立的过程。
  - ✓ 当语法分析需要读单词时，就调用“词法分析程序” 。
  - ✓ 当语法分析正确需要生成相应的目标代码时，则调用“代码生成程序”。
- 用表格管理程序建立变量、常量和过程标识符的说明与引用之间的信息联系。
- 出错处理程序：对词法和语法分析遇到的错误，给出在源程序中出错的位置和错误性质。
- 类 p-code 程序：一种假想栈式计算机的汇编语言

20. PL/0 语言

PASCAL 语言的子集，功能简单，结构清晰，可读性强，具备了一般高级语言的必备部分。

PL/0 源程序 例1：

```
const a=10;
var   b,c;
   procedure p;
   begin
        c:=b+a
   end;
begin
   read(b);
   while b#0 do
        begin
                call p;
                write(2*c);
                read(b)
        end
end.
```

21. 语法描述的目的：用有穷的文法形式，描述（验证）无穷的句子形式。

22. 语法描述的常用形式：
- 非形式化描述：不规范、不常用

> 非形式化描述形式：
> 1）任何标识符是表达式
> 2）任何整数是表达式
> 3）表达式1+表达式2仍是表达式
> 4）表达式1*表达式2仍是表达式
> 5）（表达式1）仍是表达式
> ⋮

- 语法描述图：直观、易读
- EBNF（扩充的巴科斯-瑙尔范式）

  BNF（BACKUS-NAUR FORM）是根据美国的 John W.Backus 与丹麦的 Peter Naur 来命名的，它从语法上描述程序设计语言的元语言。

  采用 BNF 可说明哪些符号序列是对于某给定语言在语法上有效的程序。

  EBNF 的元符号含义：

< >	表示语法构造成分（语法单元）
::=	定义为
|	或
{}	表示花括号内的语法成分可重复
[]	表示方括号内的语法成分为任选项
()	表示圆括号内的成分优先

23. PL/0 语言语法的 EBNF 描述（P13）

<程序>::=<分程序>
<分程序>::=[<常量说明部分>][<变量说明部分>]
[<过程说明部分>]<语句>
⋮
<赋值语句>::=<标识符>:=<表达式>
<表达式>::=[+|-]<项>{<加法运算符><项>}
⋮
<项>::=<因子>{<乘法运算符><因子>}
<因子>::=<标识符>|<无符号整数>|'('<表达式>')'
<加法运算符>::=+|-
<乘法运算符>::=*|/
⋮
<标识符>::=<字母>{<字母><数字>}
<无符号整数>::=<数字>{<数字>}
<字母>::=a|b|c|...|z|A|B|C|...|Z
<数字>::=0|1|2|...|9

> **< > 内的语法单元为非终结符，其集合记为V_N**
>
> **没有< >的语法单元为终结符，其集合记为V_T**

【举例】用 EBNF 描述“整数”的例子

【例 1】

<整数> ::= [+|-]<数字>{<数字>}

<数字> ::= 0|1|2|3|4|5|6|7|8|9

利用例1的规则，推导分析“012”是否为整数

<整数> ⇒ <数字>{<数字>}
⇒ 0{<数字>}
⇒ 0<数字><数字>
⇒ 01<数字>
⇒ 012

因此，012是整数

【例 2】

<整数> ::= [+|-]<非零数字>{<数字>}|0

<非零数字> ::= 1|2|3|4|5|6|7|8|9

<数字> ::= 0|1|2|3|4|5|6|7|8|9

利用例2的规则，推导分析“012”是否为整数

<整数> ⇒ <非零数字>{<数字>}

继续推导不出0

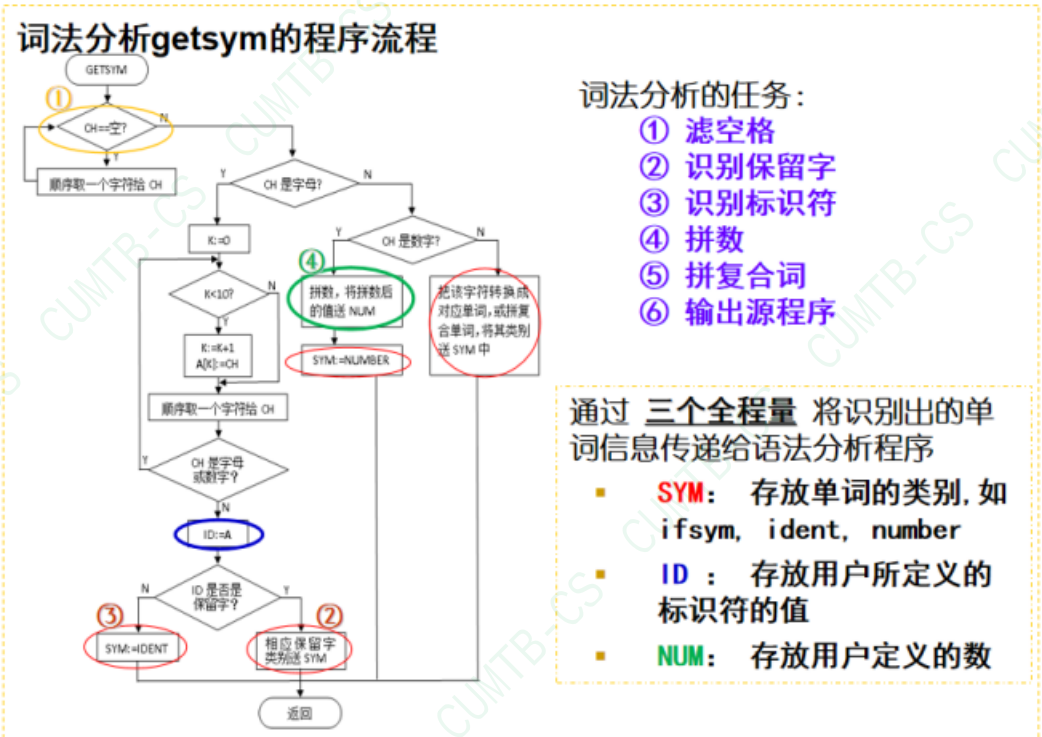
因此，012不是整数

24. PL/0 编译程序的词法分析

词法分析需要识别的单词：

- 关键字： 如 BEGIN、END、IF、THEN 等
- 标识符： 用户定义的变量名、常数名、过程名
- 常数： 如 10、25、100 等整数
- 运算符： 如 +、-、*、/、：=、#、>=、<= 等
- 界符： 如 ，．；（ ）等

词法分析过程 getsym 所要完成的任务：滤空格、识别保留字、识别标识符、拼数、拼复合词、输出源程序 （程序 P42-43）



25. PL/0 编译程序的语法语义分析

语法分析的任务：识别由词法分析给出的单词符号序列在“结构”上是否符合给定的“语法规则”。

语法分析的过程：从读入第一个单词开始，由 VN “程序”出发，沿语法图箭头或 EBNF 进行分析：

- 遇到非终结符，调用相应的处理过程
- 遇到终结符，判断当前单词是否与该终结符匹配
  - ✓ 若匹配：执行语义程序，翻译成目标代码
  - ✓ 若不匹配：报错

【举例】

语法规则（i表示任一标识符，d表示任一常数）：

① <赋值语句> ::= i=<表达式>

② <表达式> ::= [+|-]<项>{<加法运算符><项>}

③ <项> ::= <因子>{<乘法运算符><因子>}

④ <因子> ::= i | d

⑤ <加法运算符> ::= + | -

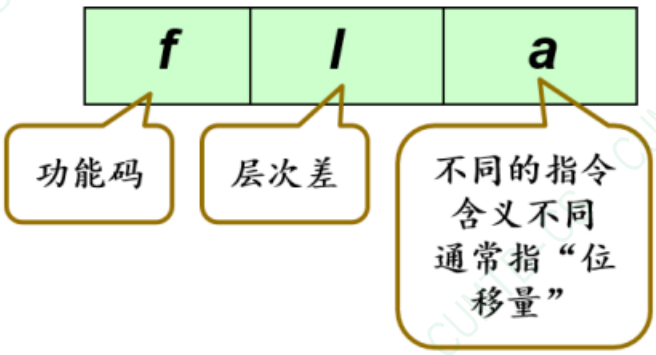
⑥ <乘法运算符> ::= * | /

分析句子 i1 = i2 + 7

<赋值语句> ⇒ i=<表达式>
⇒ i=<项><加法运算符><项>
⇒ i=<因子><加法运算符><项>
⇒ i=i<加法运算符><项>
⇒ i=i+<项>
⇒ i=i+<因子>
⇒ i=i+d

26. 目标代码 p-code：是一种假想栈式计算机的汇编语言。

- 指令格式



- 指令

LIT 0 a	将常量值取到运行栈顶
LOD 1 a	将变量值放到栈顶
STO 1 a	将栈顶内容送入某变量单元中
CAL 1 a	调用过程
INT 0 a	在运行栈中为被调用的过程开辟a个单元的数据区
JMP 0 a	无条件跳转至a地址
JPC 0 a	条件跳转，当栈顶布尔值非真则跳转至a地址，否则顺序执行
OPR 0 0	函数调用结束后的返回
OPR 0 1	栈顶元素取反
OPR 0 2	次栈顶与栈顶相加，退两个栈元素，相加值进栈
OPR 0 3	次栈顶减支栈顶
OPR 0 4	次栈顶乘以栈顶
OPR 0 5	次栈顶除以栈顶
OPR 0 6	栈顶元素的奇偶判断
OPR 0 7	
OPR 0 8	次栈顶与栈顶是否相等
OPR 0 9	次栈顶与栈顶是否不等
OPR 0 10	次栈顶是否小于栈顶
OPR 0 11	次栈顶是否大于等于栈顶
OPR 0 12	次栈顶是否大于栈顶
OPR 0 13	次栈顶是否小于等于栈顶
OPR 0 14	栈顶值输出至屏幕
OPR 0 15	屏幕输出换行
OPR 0 16	从命令行读入一个输入置于栈顶

- 生成的目标代码顺序放在数组 CODE 中。

- 目标代码举例（参见 P16）

源代码	目标代码	
const a=10;	int 0 3	在栈顶为过程p开辟3个单元的数据区
var b,c;	lod 1 3	将变量值3 入栈顶（假设b=3）
procedure p;	lit 0 10	将常量值 10 入栈顶
begin	opr 0 2	将栈顶值与次栈顶值相加，退两个栈，结果入栈
c:=b+a;	sto 1 4	将栈顶内容送入指定变量单元（c所在的单元）
end;	opr 0 0	过程p调用结束后返回

27. PL/0 编译程序的语法错误处理

(1) 语法错误

- 对于易于校正的静态语法错误，如丢了逗号，分号等

处理：指出出错位置，加以校正，继续进行分析。

- 对于难于校正的静态语法错误
  处理：给出错误的位置与性质，跳过后面的一些单词，直到下一个可以进行正常语法分析的语法单位。

(2) 语义错误
- 若标识符未加说明就引用；
- 或引用与说明的属性不一致。
  处理：只给出错误位置和性质，编译器继续工作。

(3) 运行错误
- 对运行错误，如下标越界、数据溢出等。
  编译器无法指出源程序的出错位置。

【举例】C 语言下标越界

```
int   data[10];
for(int i=1; i<=10; i++)
      data[i] = i;
```

【举例】数据溢出（注：int -2147483648～+2147483647）

```
int   i;
i = 2147483647;
i++;
```

【本章小结】

1. 编译程序的功能：将……翻译成……
2. 编译过程：
   词法分析、语法分析、语义分析、（中间代码生成、代码优化）、目标代码生成。
3. 解释程序的工作模式：
   一个个获取、分析并执行源程序语句。
4. 编译程序与解释程序的根本区别：
   是否生成目标代码。
5. PL/0 编译系统的构成：
   以语法语义分析程序为核心,词法分析程序和代码生成程序都作为一个独立的过程被语法语义分析程序调用。
6. PL/0 的语法描述：EBNF
7. PL/0 的目标代码：p-code
8. PL/0 的出错处理：语法错误、语义错误、运行错误

【课堂练习】

(1) 程序语言一般分为（A）和（C）两大类。其中（A）与人类自然语言比较接近，（C）又称为面向机器的语言。

A 高级语言
B 专用程序语言
C 低级语言
D 通用程序语言

(2) 面向机器的语言是指（ ①B ），其特点是（ ②D ）。

① A. 用于解决机器硬件设计问题的语言
B. 特定计算机系统所固有的语言
C. 各种计算机系统都通用的语言
D. 只能在一台计算机上使用的语言

② A. 程序执行效率低，编写效率低，可读性差
B. 程序执行效率低，编写效率高，可读性强
C. 程序执行效率高，编写效率高，可读性强
D. 程序执行效率高，编写效率低，可读性差

(3) 编译程序是将（B）翻译成（D）；汇编程序是将（A）翻译成（C）。

A.汇编语言程序
B.高级语言程序
C.机器语言程序
D.汇编语言程序 或 机器语言程序
E.汇编语言程序 或 高级语言程序

(4) 编译程序的工作过程可以划分为（ ① ）等六个阶段，同时还伴有（ ② ） 和（ ③ ）。

①词法分析、语法分析、语义分析、中间代码生成、代码优化、目标代码生成。
②表格管理
③出错处理

(5) 编译程序可以发现源程序的全部 （C） 错误和部分 （B） 错误。

A.语用　　B. 语义　　C. 语法　　D. 运行

(6) 要在某台机器上为某种语言构造一个编译程序，必须掌握的内容有（B C E）。

A.汇编语言　B.源语言
C.目标语言　D.程序设计方法学
E.编译方法　F.测试方法　　G.机器语言

(7) 一个编译程序，不仅包含词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成，还应包括（表格处理和出错处理）。

其中（中间代码生成）和（代码优化）不是每个编译程序都必需的。
词法分析器用于识别（单词）。
语法分析器可以发现源程序中的 （语法错误）。

(8) 程序语言的语言处理程序是一种（ ①A ）。

（ ②B ）都是程序语言的处理程序，两者的主要区别在于（ ③D ） 。

① A.系统软件 B. 应用软件
C.实时软件 D. 分布式系统软件

② A. 高级语言程序 和 低级语言程序
B. 解释程序 和 编译程序
C. 编译程序 和 操作系统
D. 系统程序 和 应用程序

③ A. 单用户与多用户的差别
B. 对用户程序的查错能力不同
C. 机器执行的效率不同
D. 是否生成目标代码

(9) 编译器必须完成的工作有（A B C F） 。

A. 词法分析
B. 语法分析
C. 语义分析
D. 中间代码生成
E. 代码优化
F. 目标代码生成

解释：因为代码优化是为了提高目标程序的质量，不是必须的，没有优化源程序一样能够转化为目标代码。而中间代码生成是为代码优化服务的，没有代码优化的编译器可以直接生成目标代码。

(10) 编译时，语法分析器的任务包括（ B C D ） 。

A. 分析单词是怎样构成的
B. 分析单词串是如何构成语句和说明的
C. 分析语句和说明是如何构成程序的
D. 分析程序的结构

(11) 与编译程序相比，解释程序通常缺少（ ①B E ），同时，解释程序（ ②D ），它处理语言时采用的方法是（ ③B ）。

① A.中间代码生成 B.目标代码生成
C.词法分析 D.语法分析 E.代码优化

② A.比较简单，可以移植性好，执行速度快
B.比较复杂，可以移植性好，执行速度快
C.比较简单，可以移植性差，执行速度慢
D.比较简单，可以移植性好，执行速度慢

③ A. 源程序语句被直接解释执行
B. 将源程序逐句转化成中间代码，解释执行
C. 先将源程序解释转化为目标代码，再执行
D. 以上方法都行

(12) 判断：“含有代码优化的编译器的执行效率高”。

错。

【讨论：为什么“含有代码优化的编译器的执行效率不一定高”？】含有代码优化的编译器，其优化是指对生成的目标代码进行了优化，而不是编译器本身得到了优化，所以，它提高的是目标代码的执行效率，而不是编译器本身的执行效率。

(13) 判断：“解释方式与编译方式的区别在于解释程序对源程序没有真正进行翻译”。

错。编译方式和解释方式实际上都进行的翻译，只是编译相当于笔译，而解释相当于口译。

解释方式下，不将于源程序彻底翻译成目标代码，而是每读入一条语句，将其翻译成中间代码，解释其含义并执行，然后再读入下一条语句，再翻译执行。

编译方式和解释方式的根本区别在于“是否生成了目标代码”。

课后 2. 编译程序有哪些主要构成成分？各自的主要功能是什么？

课后 3. 什么是解释程序？它与编译程序的主要不同是什么？

课后 4. 对下列错误信息，请指出可能是编译的哪个阶段报告的？

（1）else 没有匹配的 if;

（2）数组下标越界；

（3）使用的函数没有定义；

（4）在数中出现非数字字符。

答：（1）语法分析阶段

（2）语义分析阶段

（3）语义分析阶段

（4）词法分析阶段