

计算机程序设计语言及编译

类似于数学定义或 自然语言的简洁形式

- ▶接近人类表达习惯
- 入不依赖于特定机器

> 编写效率高

高级语言 (High Level Language)

引入助记符

- ▶依赖于特定机器, 非计算机专业人员 使用受限制
- 汇编语言 (Assembly Language)

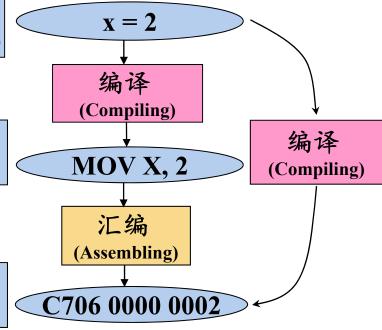
机器语言

(Machine Language)

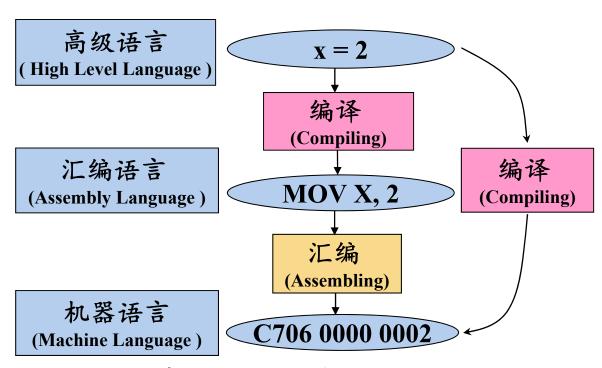
> 编写效率依然很低

可以被计算机直接理解

- ▶ 与人类表达习惯 相去甚远
- > 难记忆
- ▶ 难编写、难阅读
- ▶易写错



计算机程序设计语言及编译

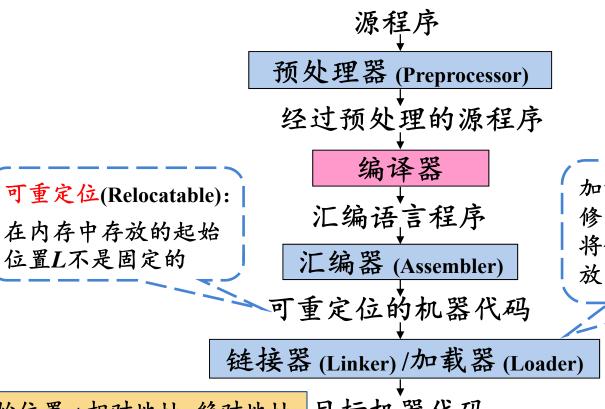


编译:将<u>高级语言</u>翻译成<u>汇编语言或机器语言</u>的过程 源语言 目标语言

编译器在语言处理系统中的位置

源程序

编译器在语言处理系统中的位置



加载器:

修改可重定位地址; 将修改后的指令和数据 放到内存中适当的位置

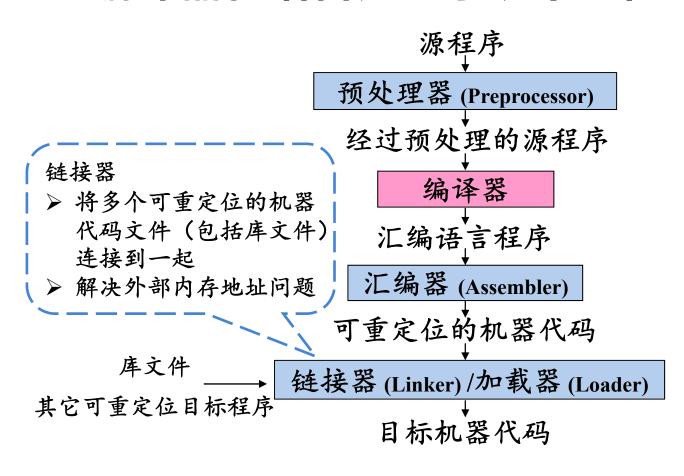
起始位置 +相对地址=绝对地址

在内存中存放的起始

位置L不是固定的

目标机器代码

编译器在语言处理系统中的位置







编译系统的结构

```
al, [31]
#include<stdio.h>
                                                                            mov al,
int main()
                                                                            call sc
                                                                           ret
                                                                      67 storechr endp
   int a,b,ge,shi,bai,m,n,i,number;
   printf("请输入(输入完两个数按一次回车键): \n");
   scanf("%d %d",&a,&b);
    while(a!=0,b!=0)
                                                                               cx, 00h
       scanf("%d %d",&a,&b);
                                                                               dx, 184fh
    if(a>=100,b<=999)
                                                                               10h
                                                                            int
       if(a>b)
            m=b,n=a;
                                                                            crt endp
                                                                               行,入口参数,行数 no
       else
                                                                               proc near, no:word
            m=a,n=b;
                             机器是怎么翻译的?
                                                                                cx, no
       for(i=m;i<=n;i</pre>
                                                                                 --0
                                                                                 cx, 1
           bai=i/100;
                                                                            .endif
                                                                            .repeat
            shi=(i%100)/10;
                                                                                 ah, O2h
            qe=i%10;
           if(i==bai*bai*bai+shi*shi*shi+qe/
           printf("%5d",i);
                                                                                 ah, O2h
                                                                                 dl,
            number++;
                                                                                 21h
           printf("\n");
                                                                            .untilcxz
                                                                            ret
       if(number==0)
                                                                      92 nextlies endp
            printf("no\n");
                                编译器
```

高级语言程序

汇编语言程序/机器语言程序

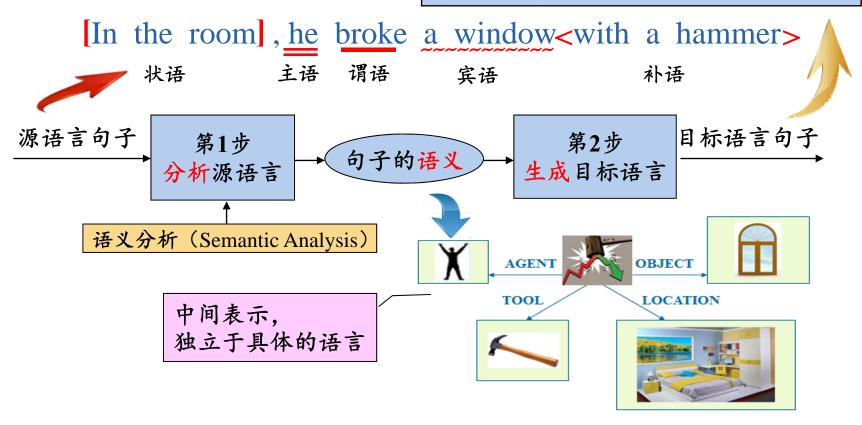
call sc

call sc

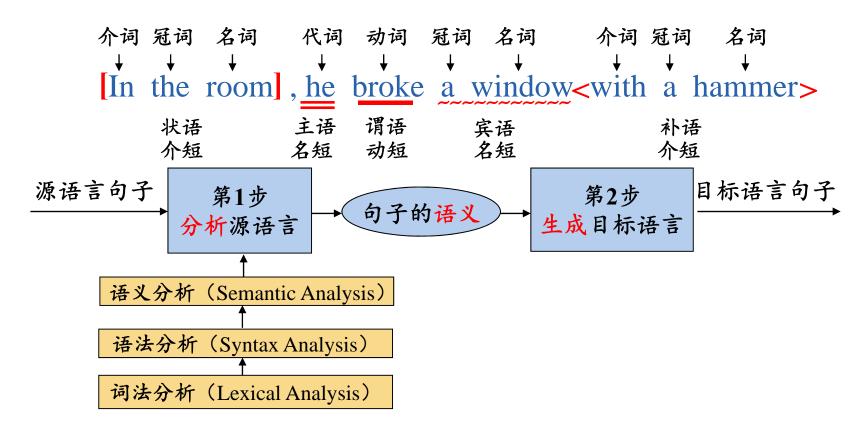
mov al, [si - 1]

人工英汉翻译的例子

在房间里,他用锤子砸了一扇窗户。

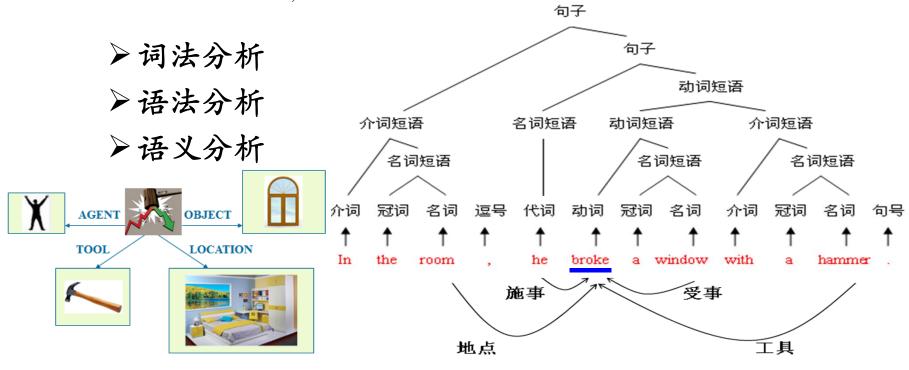


人工英汉翻译的例子



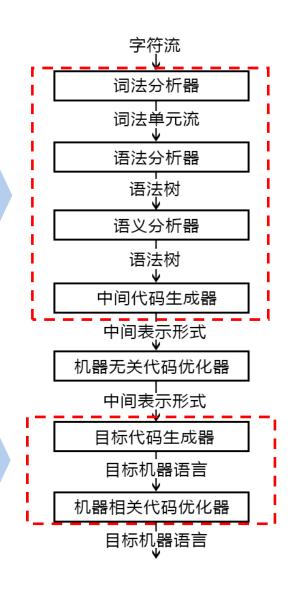
人工英汉翻译的例子

In the room, he broke a window with a hammer



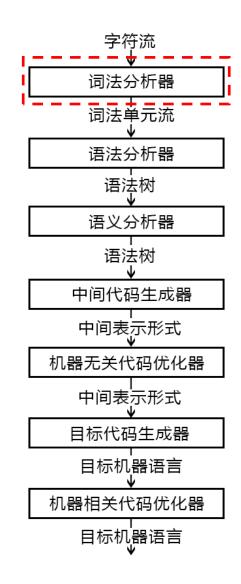
分析部分/ 前端(front end): 与源语言相关

综合部分/ 后端(back end): 与目标语言相关









词法分析/扫描(Scanning)

▶词法分析的主要任务

从左向右逐行扫描源程序的字符,识别出各个单词,确定单词的类型。 将识别出的单词转换成统一的机内表示——词法单元(token)形式

token: <种别码, 属性值>

	单词类型	种别	种别码
1	关键字	program, if, else, then,	一词一码
2	标识符	变量名、数组名、记录名、过程名、	多词一码
3	常量	整型、浮点型、字符型、布尔型、	一型一码
4	运算符	算术 (+ - * / ++) 关系 (> < == != >= <=) 逻辑 (& ~)	一词一码 或 一型一码
5	界限符	; () = { }	一词一码

例:词法分析后得到的token序列

```
▶输入 while(value!=100){num++;}
▶输出
      1
         while
               < WHILE,
                < SLP
       3 \text{ value } < IDN
                      , value
        !=
               < NE
       5 100
               < CONST,
                          100
                  SRP
                 LP
                  IDN
         num
                          num
                < INC
          ++
      10
                < SEMI
      11
                   RP
                <
```

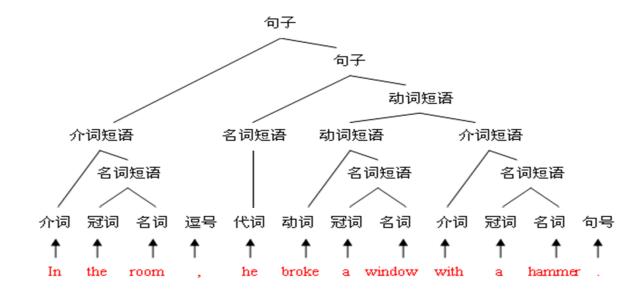






语法分析 (parsing)

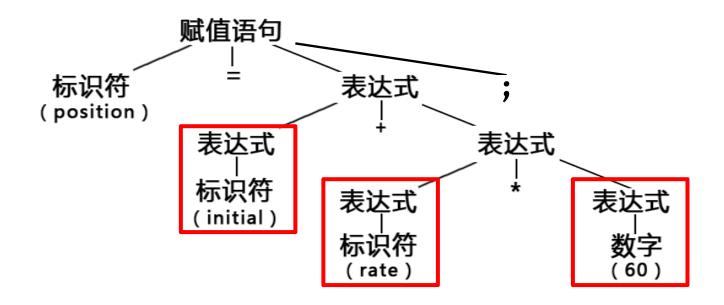
- ▶ 语法分析器(parser)从词法分析器输出的token序列中识别出各类短语,并构造语法分析树(parse tree)
 - > 语法分析树描述了句子的语法结构



例1:赋值语句的分析树

position = initial + rate * 60;

<id, position> <=> <id,initial> <+> <id, rate> <*> <num,60> <;>



例2:变量声明语句的分析树

〉文法:

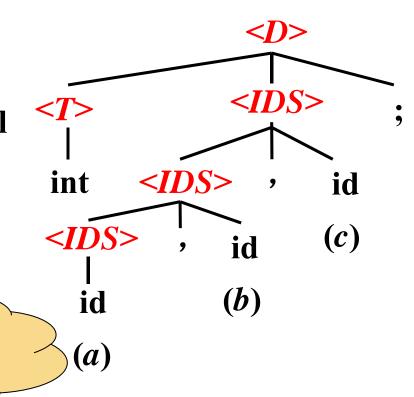
 $< T > \rightarrow int | real | char | bool$

$$\langle IDS \rangle \rightarrow id \mid \langle IDS \rangle$$
, id

>输入:

int a, b, c;

如何根据语法规则为 输入句子构造分析树?









- > 收集标识符的属性信息
 - ▶种属 (Kind)
 - ▶简单变量、复合变量(数组、记录、...)、过程、...

- > 收集标识符的属性信息
 - ▶种属 (Kind)
 - ▶类型 (Type)
 - ▶整型、实型、字符型、布尔型、指针型、...

> 收集标识符的属性信息

- ▶种属 (Kind)
- ▶类型 (Type)
- >存储位置、长度

例:

begin

real x[8]; integer i, j;

• • • • •

end

			8	x[1]
.度				
				•••••
	名字	相对地址	56	<i>x</i> [7]
	x	0	64	i
	i	64	68	<u>.</u>
	\dot{J}	68		J

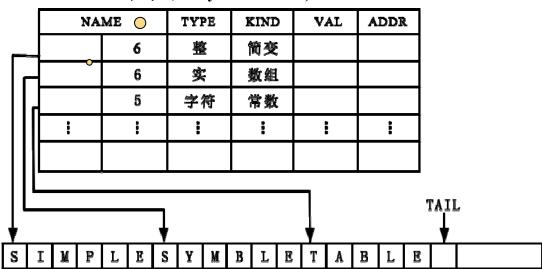
x[0]

- > 收集标识符的属性信息
 - ▶种属 (Kind)
 - ▶类型 (Type)
 - ▶存储位置、长度
 - ▶值
 - >作用域
 - > 参数和返回值信息
 - >参数个数、参数类型、参数传递方式、返回值类型、...

符号表中为什么要设计字符串表这样一种数据结构?

符号表(Symbol Table)

- > 收集标识符的属性信息
 - ▶种属 (Kind)
 - ▶类型 (Type)
 - ▶存储位置、长度
 - ▶值
 - >作用域
 - > 参数和返回值信息

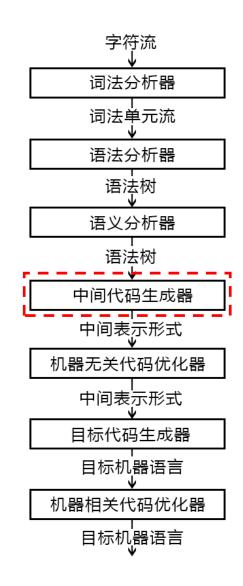


符号表是用于存放标识符的属性信息的数据结构

- > 收集标识符的属性信息
- 户语义检查
 - > 变量或过程未经声明就使用
 - > 变量或过程名重复声明
 - > 运算分量类型不匹配
 - > 操作符与操作数之间的类型不匹配
 - > 数组下标不是整数
 - > 对非数组变量使用数组访问操作符
 - > 对非过程名使用过程调用操作符
 - > 过程调用的参数类型或数目不匹配
 - > 函数返回类型有误







常用的中间表示形式

- ▶三地址码 (Three-address Code)
 - ▶三地址码由类似于汇编语言的指令序列组成, 每个指令最多有三个操作数(operand)
- ▶语法结构树/语法树 (Syntax Trees)

常用的三地址指令

序号	指令类型	指令形式
1	赋值指令	x = y op z $x = op y$
2	复制指令	x = y
3	条件跳转	if x relop y goto n
4	非条件跳转	goto n
5	参数传递	param x
6	过程调用	call p, n
7	过程返回	return x
8	数组引用	x = y[i]
9	数组赋值	x[i] = y
10	地址及 指针操作	x = & y $x = & y$ $x = & y$ $x = & y$

地址可以具有如下形式之一

- > 源程序中的名字 (name)
- ➤ 常量 (constant)
- > 编译器生成的临时变量(temporary)

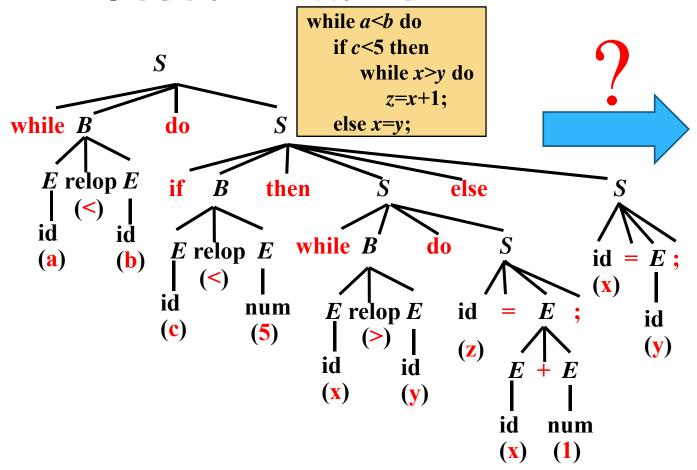
三地址指令的表示

- ▶四元式 (Quadruples)
 - \triangleright (op, y, z, x)
- ▶三元式 (Triples)
- ▶间接三元式 (Indirect triples)

三地址指令的四元式表示

```
> x = y \text{ op } z
                     (\mathbf{op}, y, z, x)
> x = op y
                     ( \mathbf{op} , y, \underline{}, \underline{}, x )
                      ( = , y, _{-}, x)
> x = y
\geq if x relop y goto n(\text{relop}, x, y, n)
                                                    AGENT
                                                                OBJECT
\triangleright goto n
                     (goto, \_, \_, n)
\triangleright param x
                     (param, \_, \_, x)
                                                    TOOL
                                                                LOCATION
  call p, n
                     ( call , p , n, \_)
  return x
                     (return, \_, \_, x)
> x = y[i]
                      (=[], y, i, x)
  x[i] = y
                      ( [] = , y , x , i )
                     ( & & , y , \_, x )
> x = &y
                                                三地址指令序列唯一确定了
                      (=^*, y, \_, x)
  x = *y
                                                运算完成的顺序
                        *= , y , _ , x )
  *x = y
```

中间代码生成的例子



```
100: (j <, a, b, 102)

101: (j, -, -, 112)

102: (j <, c, 5, 104)

103: (j, -, -, 110)

104: (j >, x, y, 106)

105: (j, -, -, 100)

106: (+, x, 1, t_1)

107: (=, t_1, -, z)

108: (j, -, -, 104)

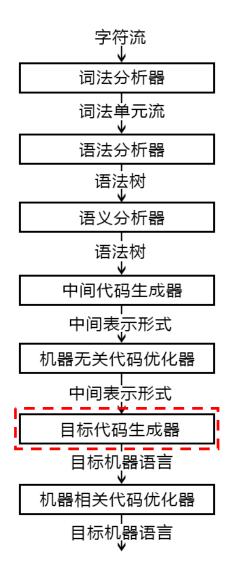
109: (j, -, -, 100)

110: (=, y, -, x)

111: (j, -, -, 100)

112:
```

- 》目标代码生成以源程序的中间 表示形式作为输入,并把它映 射到目标语言
- 》目标代码生成的一个重要任务 是为程序中使用的变量合理分 配寄存器



- > 代码优化
 - 》为改进代码所进行的等价 程序变换,使其运行得更 快一些、占用空间更少一 些,或者二者兼顾

