1. 编译过程 scc语言 入口点\_entry mainCRTStartup

其实静态库就可以看作是头文件 .h 里面包含各种函数

静态语言 变量的内存地址和大小都是固定的，（堆可能除外） 不对，是类型确定

动态语言 ：变量的内存不固定

强 弱 可否变化？ 编译型解释型

底层区别： 运行时检查类型 编译时检查类型 内存分配时机

1、由定义形成树，然后层层遍历 ，遍历中可以进行各种操作 子节点返回值经过符号处理得到父值

层层抽象之后 最底层的是什么？

一边抽象一边执行？ 语法树需要往返操作，执行完一条分支再往下一条，在该过程进行代码执行可行吗？ Eval? 转为中间代码，虚拟机解释（中间代码可以保存为文件，或者是内存中）

虚拟机寄存器 栈 计数器 内存 堆 简化为字符串数组 对象数组

语言是上下文无关的

逆波兰，寄存器可看成特殊 的栈

虚拟机是如何执行的呢？ 利用循环来分配数组就够了 if(指令=)则 ax= bx= cx= ss= memory[100]=

并不需要特别的执行，程序的本质就是处理数据， 存取改删已经在内存中进行了 哪个标识符a b c分配哪个内存也规定了（？）

编译成中间

唯独在硬件层面可能需要

只要使用原语言的api 代替就行了 如何实现自己的标准库，或者借用别人的标准库，标准库需要编译吗？ 鼠标库 ，键盘库

虚拟机的指令如何执行呢？ Add 这些直接就可以用java或者c来解释， 内存视为数组，寄存器视为变量 操作符很容易add cmp

Read指令 next()下一条指令 execute() swith 无限循环读取和执行，读取速度？

如果调用系统函数（就是c标准库函数） 可以把系统函数设置为特殊指令，比如int

记住把返回值放回寄存器

Execute中 当指令为int时 调用某个标准函数 比如print

可以保存符号表，方便外部取出变量地址，可以试图停止和恢复虚拟机运行

语言是需要标准库api的，不然连最基本的print都实现不了.，当然转换别的语言就可以了，标准库一般需要依赖平台底层，调用系统api，系统api是c写的，但是可以编译成机器码形式

本书用到的 函数 常量 变量分类 以及调用关系

TK plus+ minus- star \* divide / mod% eq== neq!= lt< leq<= gt> geq>=

Assign = dot . and& ( openpa closepa ) openbr[ closebr] 整数cint ‘’ cchar

“” cstr eof

KW 关键字 char short int void struct if else for continue break return sizeof cdecl stdcall pack begin{ end } semicolon ; 标识符 tk\_inden+n

,

Comma ,

Type( ;|声明函数体|声明[=initiallizer]{,声明 [=initiallizer]}; )

编码结构 e\_TokenCode{}

动态字符串 DynString { count capacity data} init free reset relloc chcat 增加一个

动态数组 DynArray { count capacity data } add init free search

哈希函数 elf\_hash()

单词 TkWord{tkcode,next,spelling,sym\_struct,sym\_identifier }

单词表 tkTable （数组） 单词哈希表tk\_hashtable()

全局变量 token当前取的token

Getch getc(fin)

Get\_token

文法描述图，对于每个节点都可以描述， 入口，出口 分支，重复

Declarator()

Funcbody() initializer() skip() 类型区分符type\_specifier() struct\_specifier() struct\_declaration\_list() struct\_declaration() function\_calling\_convention()

Declarator()声明符，需要等价转换

struct\_member\_alignment() 直接声明符direct\_declarator() direct\_declarator\_postfix()

形参类型表 parameter\_type\_list() funcbody()

Initializer()初值符 assignmen\_expression() statement（）语句 if\_statement() break for continue return expression\_statement() compound\_statement()复合语句

Expression（）表达式 赋值表达式assignment\_expression 非等价变换？语义本身就有不合理时，增大不合理范围，然后语义分析时判断 equality\_expression relational\_expression

Additive\_expression multiplicative\_expression() unary\_expression() 一元

Sizeof\_expression() postfix\_expression()后缀表达式 primary\_expression()初值表达式

实参表达式表 argument\_expression\_list()

语法状态：syntax\_state 缩进级别 syntax\_level

状态值：SNYX\_ NUL空状态 SP空格 SNTX\_LF\_HT换行并缩进 声明、定义、语句结束时

在语法分析的过程中，调整各状态

然后每次调用get\_token() 都执行syntax\_indent()

Colortoken() 染色 lex\_state词法状态NORMAL SEP 设置控制台文本颜色（获取输出句柄）

print\_tab() printf(token)

token是当前符号的编码

语法分析，循环取token 判断接下来的语法是否符合语法

多个产生式如何选择 让选择之间完全分离，不要有重复情况 first集select集

一、词法分析 单词 编码 分类 关键字、常量、标识符、运算符、分隔符

Int return “hello” 0 () printf

不同词语着色

语法分析 是否是文法的正确 语法缩进

语义分析 生成obj 符号表 错误处理

链接 生成exe 库文件

Vc6中是用cl.exe编 link.exe链接

文法： ::= 如何定义语言

语法 ：记号组合规律 语义：记号含义 ， 符合语法可能语义很荒谬 花咬鱼

句子 主 谓 宾 名 动 名词：：=”你”|”我”

字母表 关键字：if else for 运算符：+ -\* 、 分隔符; 标识符 ：字母数字下划线（不能是关键字）

字母表 语法字母表:所有单词 词法分析，所有字符

符号 单词或字符

符号串 语法int a=1; 词法：int 有顺序

符号串集合

相等串 连接 乘积 幂 闭包运算

文法： 列出句子的集合 VN非终结符可以取代（分解，继续定义 ） VT终结符（通常是单个符号） P产生式规则 S开始符号 非终结符

文法分类 0，1，2，3（正则） 识别器 图灵 线性有界自动机 下推自动机 有限状态机 2 ：左边是非终结符 0 1 2 3是层层包含关系

巴科斯范式：bnf 从起始开始 替换前面符号的规则

定义为

或者 |

<非终>

有符号整数：：=<无符号整数>|<正负号><无符号整数> 无符号整数::=<数字>|无符号<数字> 数字::=0|1|2 正负号::=+|-

Ebnf 简化 [] 0到一次 {任意次重复} 可以是无限 “终结符” 无符号整数::={<数字>} 可以n 位 () 看成一项

词法分析：状态转换图

对各词进行定义（使用编码） ： 整数INTERGER

终结状态用双圈图

结点状态：用数字序号？ 边 上的符号是识别的词形 开始状态 接收状态

获取字符 跳过空白 switch 是字母 是数字吗？ 是+ -乘除吗？ 如果是就打上标签，给一个编码 IDENTIFIER（标识符） INTERGER整数

语法：自顶而下 从开始符号bnf 使用产生式 寻找匹配的串

自底而上 从输入串归约到开始符号 BNF

推导出和输入的单词串完全相同的句子 不确定的自顶而下：带回溯，非终结符有多个候选，穷举

确定的自顶而下，根据输入符号选择某个产生式

Ll分析器 从左往右扫描输入串 l用最左推导 LL（k）需要向前看k个符号才能决定如何推导 使用哪个产生式

LL（1） LL（0）当没有| {}等的时候，只有一个替换项

LL（1） 只用看一个字符（？）

First集 首字符集

Follow集

Select集

判断是否LL（1）文法

1、递归子程序法 2、预测分析法

每个非终结符写个递归 识别推导的串

Program statement expression

递归向下获取字符 get\_token(此时的仍然是单字符？) if progrm()->statement()->expression()->

报错

非LL（1）形式等价变换 消除左递归

1. SC语言

C89 字母 数字 空格 标点 特殊字符 字符常量可用汉字 源码字符集，执行字符集（转义）

基本字符集 扩展字符集（只能字符串中或注释中）

字母 a-z A-Z 下划线 数字0-9 标点+-\*/ <> . & ; 空白 制表换行 空字符（/0 字符串终结）

词法分析 关键字 char int short void struct if else break return

语义：分类 ：数据类型 控制语句 类型长度 调用约定

名称

含义：声明字符型变量或函数 退出循环

标识符:=<非数字>{数字|非数字} 第一个字符必须是数字或者下划线 语义 ：变量名 函数 名 存储类型和数据类型 作用域

整数常量

字符常量::= 转义序列 |所有字符 字符串常量

运算符 <大于号> 分隔

注释

2、语法定义 约束条件，并不是所有符合情况的都成立，**需要特别指出特例，**方便编译报错 其实语法定义不一定能覆盖所有的情况，简单的程序也不需要那么多情况，只要基本覆盖就行了

层层下降 可能有1个或多个分支，

树怎么划分合理呢？ 等价替换？ 层次应该多还是少？

其实下面的划分也未必合理，可以自己规划

原则上是解析出比较明确的小块 然后解析成中间代码 子树的有无、子树的重复、多选一

外部声明 声明 |函数定义

函数定义 type 声明符 函数体 int a() {}

函数体 复合语句 { }

声明 类型区分符[初值申明符表] 分号 type [初值声明符表] ; int a=1

初值申明符表 初值声明符{，初值声明符} a,b,c a=1,b=1,c=1

初值声明符 声明符|声明符=初值符 a[1] a= b=c

类型区分符 int|void|char|short|结构区分

结构区分符struct indent { 结构生明表 } | struct 标识符

结构声明表 结构声明{结构声明}

结构声明 类型

结构声明符表 声明符{,声明符} int a,char b[],int c(a,b)

声明符 {\*}[调用约定][结构成员对齐] 直接声明 \*a a()

调用约定cdecl|stdcall

结构成员对齐\_align(cint)

指针 \*

直接声明符 标识符 后缀 a[] a()

直接声明符后缀 [] [0] () (形参)

参数声明 类型 声明符 int a

初值符 赋值表达式 a=b a=1

语句 复合|if|for|break|continue|跳转

复合语句（块） “{“ {声明}{ 语句 } “}”

表达式语句 空语句 [表达式] ; 分号

;

选择语句 if(表达式)语句 [else 语句]

循环语句 for(表达式语句 表达式语句 表达式语句) 语句

跳转语句 continue; break; return 表达式; 分号

表达式 赋值表达式{，赋值}

A=b,b=c,c=d

赋值表达式 <相等表达式>| <一元>=赋值表达式

A=b=c=d sizeof(type)=(b+c+d) ??? 所以说不完全能覆盖

相等类表达式 关系{=={关系}| ！=关系 } 包括单一关系表达式

A==b

关系表达式 加减类 {>{加减类} |<加减|<= |>= } 包括0次比较

A<b<c<(d+1) a<b 注意：的值为0

加减类表达式 乘除{+乘除|-乘除} 包括0次加减

A\*b+b\*c-c/d

乘除类表达式 一元{\*一元|/一元|%一元} 包括0次乘除

A\*b/c\*d a\*a[2] \*p\*[a ]

一元表达式 后缀| &一元|\*一元|+一元|-一元 | sizeof(type)

Sizeof(int) \*a \*\*p

后缀表达式 初等 { [expression]| () | (实参表)|.indent| -> indent }

A[c+d] func() func(a,b) a.b a->b

初等表达式

indent | cint| cstr|cchar | (exprssion)

a 1 “ok” ‘a’ (a+b)

词法分析需要用到动态字符串（单词长度不限） 单词需要用动态数组保存 查找应该用哈希表（不存重复单词）

动态字符串：长度，容量，指针 初始化容量 释放 重置 重分配 追加字符

动态数组 个数 容量 数据指针（数组名） 重分配 追加元素 初始化容量 释放数组 查找元素

字符串哈希 simplehash rs js pjw elf bkdr sdbm djb crc ap

这里用elf **计算hash地址**

哈希key

单词表，其实大部分的单词已经确定了，只需要加入的是标识符

单词存储结构： TkWord 编码 next 哈希冲突的同义词 字符串 指向定义的结构 指向表示的标识符

全局变量：哈希表容量 哈希表（数组） 单词表（动态数组）

放入单词表add 计算单词哈希地址，放入哈希表的地址中 返回值是单词

单词表中查找单词： 调用哈希表查 key 返回单词 为空说明没有，不为空匹配则返回，不匹配则沿着next找

运算符、常量、关键字直接放入单词表

标识符查入单词表 先查找，找不到才插入 单词：开空间 设置next 加入哈希 加入单词表 设置编码 设置字符串拼写（=输入的单词） 最后在拼写后面加上/0

分析初始化 lex\_init： 让关键字，运算符先进入单词表 列出所有的关键字，运算符的单词结构数组【{}{}】 开启一个动态数组（单词表） 循环遍历数组 执行 直接放入单词表的步骤

编译错误处理: 错误发生在哪个文件 第几行 错误级别 错误编码C

错误处理： 级别 工作阶段

异常处理

编译警告处理warning

致命错误处理error

提示错误

主程序 取单词程序：get\_token 预处理 注释解析 空白处理 标识解析 常量解析 字符/串常量解析 先解析 如果是标识符，把解析的**单词字符串**插入到单词表 然后设置当前token= 单词编码 如果解析是运算符，不加入单词表,直接当前token=TK\_ 一直解析到EOF

Switch

预处理：忽略空白和注释 如果是’ ‘ \t \r跳过空白符 如果是/看下一个字符\* 是就注释处理 如果不是就ungetch(ch,fin)退回输出流

读取字符 getch 是否有指针？ Ch=getc(fin) 直接读出一个字符，每次执行多读一个

注释处理 中getch 然后循环getch,发现换行，文本结束，\*就 结束循环 如果是换行，行号+1 继续getch，并循环所有过程 如果\*后是/ getch并返回 如果到文件尾没看到\* 报错并返回

跳过空白符： 当 ch=’’ 或t r 时 循环 如果是r就多getch一位 是n的话行号+1 不是n就返回 如果不是r就循环getch，直到读到不是空白符

解析标识符： 字符判断： 是否字母或下划线 是否为数字 true false

解析： 重置动态字符串 当前字符加入字符串,getch 当是字母或数字时，getch 并加入字符串,最后字符串+\0

解析整数 （浮？） 源字符串 单词字符串 如果是数字就循环加入（两串同步），getch，直到非数字 如果非数字 点号的话 继续循环加入 最后+\0

Tkvalue=atoi()

解析字符串 函数需要一个边界标识’ “ sep标识 源字符串先加入sep 把原本的字符放入源字符串 转义后字符放入 单词字符串 c转义字符

循环，遇到另一个sep就停止 如果\\ 就直接加入并getch之后需要转义 0开头 a开头 b开头 t开头 等都需要 c=/a 等转义字符 如果转义字符不在范围内，默认是原样输出，但是有部分会是非法转义，错误或者警告 分别按情况加入源字符串和单词字符串

如果不是’// 就直接读入 getch循环 最后补\0

标识符分配索引码

词法着色

单词编码 结构 e\_TokenCode{ TK­\_PLUS } KW\_ 关键字 EOF文件结束符

主程序 main 读取文件 init() 然后循环get\_token 直到token=EOF 显示代码行数

Clearup()扫尾程序 关闭文件

初始化 行数=1

扫尾程序：释放单词表

全局变量 单词哈希表 单词表 单词字符串 源字符串 单词值 当前的字符ch 当前单词编码 token

二、语法分析+缩进

其实语法分析可以看出来，是从上往下的一棵树，每个语法结构可能是多选一（条件语句或者是循环语句或者是赋值语句），也可能是并列多个同级的（多个定义语句）

对语法分析的过程就是不断向下的过程，向下调用

全局变量 语法状态 是换行还是分隔 缩进级别

语法成分:文法定义，代码讲述，文法描述图

多用些TK\_ 这种作为非控制符

分号：TK\_SEMICOLON open close()pa []br

1、翻译单元 没到文件尾就循环调用 外部声明

外部声明是不符合LL（1）要等价转换

<类型区分> (<分号>|<定义><函数体>| )

连续向下调用 token gettoken 参数l SC\_GLOBAL SC\_LOCAL 解析状态局部全局

Skip函数

不断get\_token

Get\_token函数

语法缩近syntax\_indent print\_tab

Token 当前token

Main函数 获取文件 init getch get\_token 翻译单元（） clearup

Translate\_unit

external\_declaration

六 符号表 同名符号 但是作用域不同

符号属性

语义合法性检查 不能作为左值 未定义 不能相加

数据结构 ：栈 动态栈 初始化容量 push pop get\_top is\_empty free

符号表 全局符号栈 局部符号栈

符号： 编码 寄存器 关联值 数据类型 关联的其他符号next pre\_tok 前一同名符号

符号入栈 直接入栈

动态判断是全局还是还是局部，切换当前栈 然后放入

函数符号放入全局

节名称放入全局

单词：的属性 单词的标识符 sym\_indentifier sym\_struct

弹出符号 直到栈顶为?

符号查找 查找结构定义

数据类型 Type {t ref} t的值是类型 编码 0-6

全局变量：

存储类型

层层向下生成符号表

单词（存于单词表） TKWord:sym\_struct sym\_identifier

tktable

Stack base top size init push pop get\_top is\_empty destroy

Global\_sym\_stack local\_sym\_stack

Symbol: v 编码 r寄存器 c type next prev\_tok

Sym\_Direct\_push sym\_push func\_sym\_push

Var\_sym\_put() 变量 sec\_sym\_push

Sym\_search（）

返回值是

删除sym\_pop

查找 struct\_search sym\_search

Type type ：t ref : T\_int char short void ptr指 func struct btype array

Char\_pointer\_type

Int\_type

Default\_func\_type

存储类型 storageclass sc\_ global local llocal cmp valmask lval sym anom struct member params

l 是局部还是全局

语义分析，进行条件判断检查 同时sym­­\_push

Type v,hasinit,r ,addr sym 全局

Error expect(“error”)

在外部中 设置 type 传入到Type\_specifier 进行赋值type->t

Type\_specifier( type) 如果是struct 则修改type->ref

可以发现语义分析阶段，每个节点 需要设置很多参数和局部变量，这是为了对语义进行检查 如果不符合语义就报错

**定义一个指针结构并传入？ 指向结构的指针？ 直接传地址进去（相当于传匿名的指针）func(&a)**

**结构成员对齐**

Mk\_pointer()

如何获得一个函数的输出？ 1/全局变量穿透 2、传入指针参数（地址）3、return

一些特殊的类型 char \* struct \* func\*

声成符号都干了什么？

七、生成目标coff文件：文件头 节表 符号表 重定位信息

文件头 节头表 节数据/代码 coff符号表 字符串表,符号指向字符串 main printf

节 .bss .data .idata .reloc .text代码节

Data rdata 常量字符串

符号表sym .symtab

Coff 重定位 每个节都有 rel节 重定位表

机器类型

节结构 struct\_section { }

Section\_new() section\_ptr\_add() section\_relloc()

New\_coffsym\_section() coffsym\_add()

八、X86机器语言

通用指令 前缀 prefix opcode操作码 modr/m sib 偏移 立即数

操作数长度前缀 地址长度前缀 地址超越前缀（不使用默认段寄存）默认ds 吧 ，换成cs,es,fs,ss gs 串操作 重复指令 重复前缀 锁定前缀

操作码一般是一到二个字节 需要modr/m辅助

数据寄存器 变址寄存器 栈指针寄存器ebp esp 段寄存器csds es ss fs gs 指令指针寄存器eip 标志寄存器cf pf af

分类 数据传送 算术运算 逻辑运算 控制转移jmp call ret jcc 串操作 处理器控制nop

操作数栈

操作数struct push pop swap assign(赋值)

生成通用指令

生成指令前缀

生成操作码

生成Modr/m

生成操作数

生成数据传送

算术和逻辑

控制转移

调用

寄存器分配

寄存器溢出

生成机器语言

九、语义分析