

continue to be updated

编译器：自己写一个编译器。越早开始越占便宜

pointer type 到底是用哪种形式？是六十四位机的8byte 还是三十二位机器的4byte

注意：之后再不对可以尝试着注释掉输出的优化 因为t64是有一些问题 的

这边存在着问题emmm 祖传写的是4 但是在本机上测试必须使用64位才能够保证我内存申请的空间
因为本地的地址是64位的

温馨提示：

ravel请用

<https://github.com/Yveh/ravel/tree/bd8e38e0cf57dd6b1d108b224c1c4966485de96> 这个
或者 <https://github.com/stneng/ravel> (<https://github.com/stneng/ravel/tree/8f52f0ac28392042db7bc953ec06025e900b91ed>) 这个

请不要使用 <https://github.com/Yveh/ravel> (<https://github.com/Yveh/ravel/tree/a05c0b788f6a93a6ecf2ffa11261857c05432f72>) 这个 因为他真的很慢

(一)课堂笔记

9.17

计算机系统与实现

累计扣分

Mx*

parser&lexer

src->#(pre-process)

语言定义：字符 词法结构 (lexical structure)

句法结构 (syntactic) ->AST(抽象语法树)随便怎么写都行

词法分析器：词法分析 正则表达式 有限状态自动机

lexer 词法分析机

流程：

去除不必要的字符

...

字符流转换为词素流 (token)

生成式方法：正则表达式

识别式方法

NFA非确定有限状态机

NFA 确定有限状态自动机

状态机只要有一条路走通就可以了

有用的东西：无符号整数

parser(语法分析机)

文法和语言

正则文法

上下无关文法

自顶向下分析 (LL1)

自底向上分析：LR

对比：词法分析：字符 变成 词素

语法分析：词素 变成 语法树

文法和语言：

一个文法识别一个语言

9.29

cs2966 2分 semantic week7(9) codegen week18 register 23(baseline区间给分)

cs2965 2分

cpu riscv70 cache10 特权指令

toma 70 80 100

在src里面新建a 文件夹 写a.g4



(二)借鉴与学习 (日志)

一.outline(Yx ZYH repo)

传统的编译器通常分为三个部分，前端(frontEnd)，优化器(Optimizer)和后端(backEnd)。在编译过程中，前端主要负责词法和语法分析，将源代码转化为抽象语法树；优化器则是在前端的基础上，对得到的中间代码进行优化，使代码更加高效；后端则是将已经优化的中间代码转化为针对各自平台的机器代码。

基本的运行阶段：

1. 生成你的编译器 / Build your compiler : 使用系统编译器编译你的编译器代码构建你的编译器的过程。
2. 编译目标代码 / Compile target code : 使用构建的编译器编译Mx*语言，如果编译正确输出目标汇编代码，反之编译器应当以非0返回值退出。
3. 执行目标代码 / Execute target code : 使用模拟器运行你的代码的过程。

program - function and class definition - statement - expression - primary

流程

- 1.g4
- 2.The `error` class is a trivial self-implemented error type. The `YxErrorListener` is defined
- 3.Design The AST
- 4.AST Builder a visitor on the parse tree
- 5.Scope Every scope records variables defined in it.
- 6.Semantic
- 7.Brief introduction to type
- 8.Codegen (IR(Intermediate Representation))
- 9.IR design
- 10.IR builder
- 11.IR printer(传给全局)
- 12.prune
- 13.Instruction selection
- 14.Register Allocation

一篇博客：

1. **词法分析**：单词与记号、正则表达式、有限自动机、从正则表达式到有限自动机的转换、词法分析器的实现
2. **语法分析**：上下文无关文法、递归下降分析、LR 分析、错误处理、语法分析器自动生成
3. **语义分析**：类型系统、属性文法、语法制导翻译、符号表管理、抽象语法树、线性中间表示、图中间表示
4. **中间代码生成**：变量地址分配、算术表达式翻译、布尔表达式翻译、数组、结构体和字符串的翻译、控制流的翻译、函数调用的翻译
5. **目标代码优化与生成**：目标体系结构、树匹配代码生成、基于动态规划的代码生成、寄存器分配、指令调度、控制流分析数据流分析、死代码删除、常量传播、拷贝传播、静态单赋值形式

二.ast阶段任务

hz cq yxy

builder collected 变量值 作用域什么的 checker

然后是util几个工具的实现

astnode先实现 然后建树

一颗抽象语法树每个节点代表一个构造，某个节点的子节点代表对应构造有意义的组成部分。 (龙书)

```
BOOL_LITERAL: TRUE | FALSE;  
INTEGER_LITERAL: '0' | [1-9][0-9]*;  
STRING_LITERAL: '"' (~["\n\r\\"] | '\\'"nr\\")*? '"';  
NULL_LITERAL: NULL;
```

字面的基本数据类型

1.yxy:

- utils
 - error
 - errormessage(报错信息的基类)
 - scope(作用域) 先不用管?
 - symbol 先不用管?
 - type 简单可能是检查类型..?检查等号...?
- frontend 建树的函数
- ast ast节点的声明和设计

2.cq

- ast costexprnode exprnode programnode statemenode typenode
- frontend scope type entity
- util 给后续用的工具

跑yx 然后看树建造的过程

semantic check用了访问者模式来实例每一个接口

main里面有一步就是进行semantic check

```
@Override public ASTNode visitVardefStmt(YxParser.VardefStmtContext ctx) { return visit(ctx.varDef()); }  
  
public T visit(ParseTree tree) {  
    return tree.accept(parseTreeVisitor: this);  
}
```

```
public <T> T accept(ParseTreeVisitor<? extends T> visitor) { visitor: ASTBuilder@1025
    if ( visitor instanceof YxVisitor ) return ((YxVisitor<? extends T>)visitor).visitVarDef( ctx: this); visit
    else return visitor.visitChildren( ruleNode: this);
}
```

```
@Override public ASTNode visitVarDef(YxParser.VarDefContext ctx) {
    ExprNode expr = null;
    String typeName;
    if (ctx.type().Int() != null) typeName = ctx.type().Int().toString()
    else typeName = ctx.type().Identifier().toString();
    if (ctx.expression() != null) expr = (ExprNode)visit(ctx.expression());

    return new varDefStmtNode(typeName,
        ctx.Identifier().toString(),
        expr, new position(ctx));
}
```

visit parsertree的流程

ast builer 返回的每一个接口的T都是一个astnode

tostring equals 每个object都是有的

！！！ respect from the continuing 进步

```
String[][] s = new String[2][];
s[0] = new String[2];
s[1] = new String[3];
s[0][0] = new String("Good");
s[0][1] = new String("Luck");
s[1][0] = new String("to");
s[1][1] = new String("you");
s[1][2] = new String("!");
```

从高维开始分配空间

- ✓  EXPRnode
- ✓  CONSTEXPRnode
 - ↳  Constbool_ASTno...
 - ↳  Constexpr_ASTno...
 - ↳  Constint_ASTnode...
 - ↳  Constrnull_ASTnod...
 - ↳  Conststring_ASTn...
 - ↳  ArrayExp_ASTnode.java
 - ↳  BinaryExp_ASTnode.j...
 - ↳  Binary_Enum.java
 - ↳  Expr_ASTnode.java
 - ↳  Front_UnaryExp_AST...
 - ↳  FuntioncallExp_ASTn...
 - ↳  IdExp_ASTnode.java
 - ↳  LambdaExp_ASTnod...
 - ↳  MemberExp_ASTnod...
 - ↳  NewExp_ASTnode.java
 - ↳  Post_UnaryExp_ASTn...
 - ↳  Single_Enum.java
 - ↳  Thisexpr_ASTnode.java
- ✓  STATnode
 - ↳  Breakstat_ASTnode.j...

- ↳ Continuestat_ASTno...
↳ Exprstat_ASTnode.java
↳ Forstat_ASTnode.java
-

- ↳ Ifstat_ASTnode.java
- ↳ Returnstat_ASTnode....
- ↳ Stat_ASTnode.java
- ↳ Suite_ASTnode.java
- ↳ Valdeclstat_ASTnode...
- ↳ Whilestat_ASTnode.j...

TYPEnode

- ↳ Arraytype_ASTnode.j...
- ↳ Booltype_ASTnode.ja...
- ↳ Classtype_ASTnode.j...
- ↳ Inttype_ASTnode.java
- ↳ Nulltype_ASTnode.ja...
- ↳ Stringtype_ASTnode....
- ↳ Type_ASTnode.java
- ↳ Voidtype_ASTnode.ja...

VALDECLnode

- ↳ Classdecl_ASTnode.j...
- ↳ Constructdel_ASTno...
- ↳ Declare_ASTnode.java
- ↳ Fundecl_ASTnode.java
- ↳ Paralist_ASTnode.java
- ↳ Singlervaluedel ASTn...

三.semantic scope symbol

```
struct a  
{ b x; };  
struct b  
{ a x; };
```

Of course, this is not a good design in C/C++ since you can have x.x.x..... However, in Mx and Yx, all variables are in form of reference, so we can have:

```
a y;  
y.x = null;
```

And the design above is good.

What is the problem? If you check the correctness of `struct a`, you do not know if `b` is a legal type, vice versa.

To solve the problem, we use another pass named `SymbolCollector`, which briefly collects all type names into a global scope. A global scope is a class derived from `Scope` by adding a map to collect these types. See `util/globalScope` for more details.

然后这之后的YX就看不懂了

其他就看不懂了 估计是scope ? symbolcollector? 还有4, 500行代码可以过掉semantic check
scope 包括一个父类的scope 自己也包含一个hashmap or hashset

4 命名空间:

在同一个作用域里，变量，函数，和类，都分别不能同名（即变量不能和变量同名，其余同理）。如果重名视为语法错误，注意作用域规则。在作用域内，变量和函数可以重名，但是类不可以和变量、函数重名。

注意这个是在同一个作用域内，不同作用域有覆盖效果

semantic 和后期没有关系 最多帮助你了解asttree的架构

改一下expr的type 然后注意很多scope都是先new (currentscope) 最后
`currentscope=currentscope.parentScope()`

二元运算符就是判断左右两边的运算对象是不是一样 || &&等一系列东西左右只对特定类型成立

基本操作

```
currentScope = new Scope(currentScope);  
currentScope = currentScope.parentScope();
```

不用另外开type了直接用Node节点一次两用算了

`println` `printint` 啥的可以在调用函数的时候特判一下 当类型是string的时候可以不用在scope中定义即可调用

使用accept visit模式是能够较好的实现访问的一种方法，兼顾了多态

xxx.accept(this)----> 进入一个node(xxxx的数据类型) 的visit函数---->visitor.visit(this)---->调用这个node的visit函数

每一个visit函数最后都要return it;来把得到的值返回上一层

如果我只用一个scope的话最后可能里面的内容比较多而且很多东西都会在简单情况下没有被使用到，继承的话层次性会比较好，读起来会比较舒服，但缺点是构思要花费时间

我还是采用先进行类似symbolcollecter的操作，不是上来就check!

hbh不具有参考性 版本不一样

symbol只是用来作为一些记录当前处在范围内的一些基本信息

可以不用 用许多参数来代替

只要string 映射到node就行了，基本上所有的信息都存储在node之中，我只要node里面信息存的够多 我什么都不用 只用一个scope

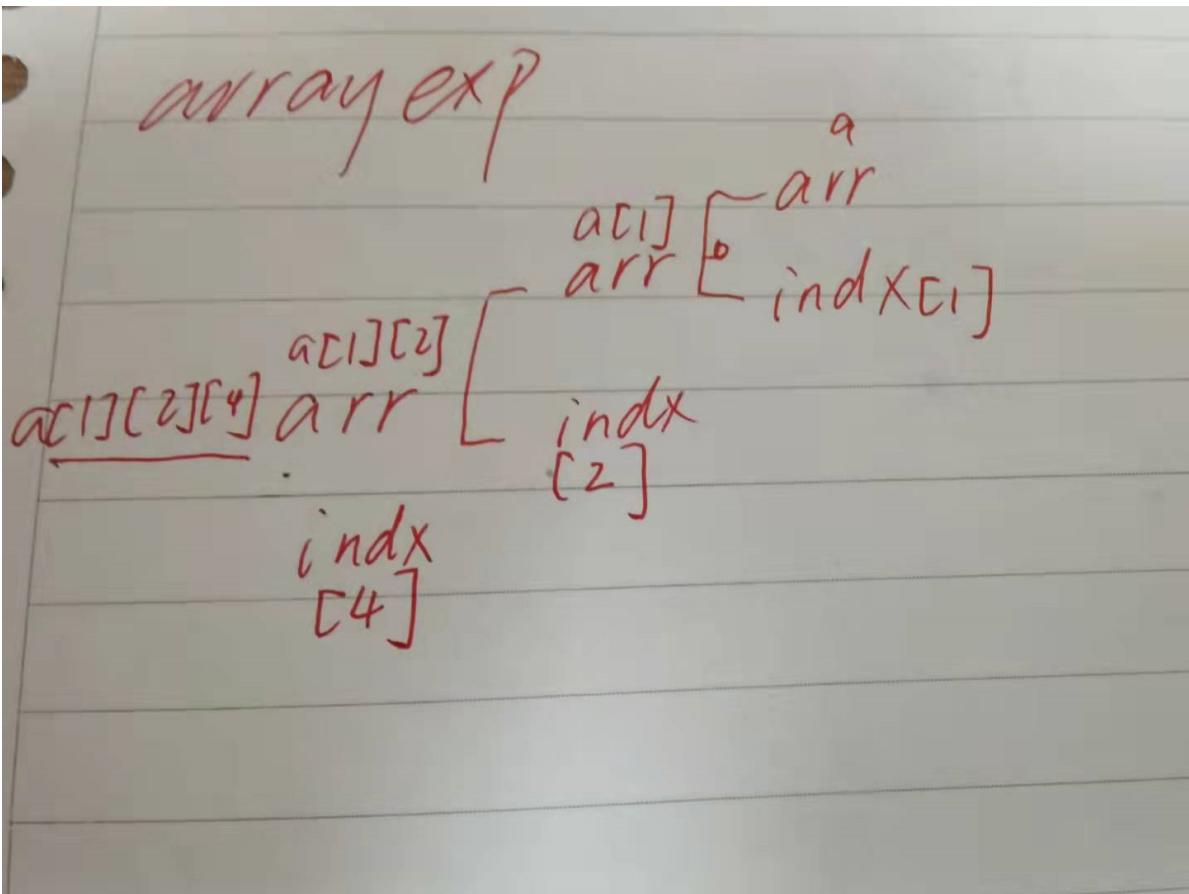
目标：先通过一个比较两边类型的自己造的数据点 (if 加上class 两次define)

不用在意开始的arraytype 所有的信息都已经处理出来了

new array 记录了有几个括号括号前面的exprlist

arrayexpr 通过下面的方法处理出了一个有层次结构的array

```
@Override  
public ASTnode visitExpr_array(MxParser.Expr_arrayContext ctx) {  
    Expr_ASTnode array = (Expr_ASTnode) visit(ctx.array);  
    Expr_ASTnode index = (Expr_ASTnode) visit(ctx.index);  
    return new ArrayExp_ASTnode(new position(ctx.getStart()), null, null,  
        array, index);  
}
```



处理数组类型

int [][] [] [] a;

a[][][]是二维的

在arrayexpr里面可以把他实现成一维的
然后就可以处理a[][]=1等一系列问题

scope考虑和上一个scope的关系

这回todo流到了194行

a.foo()

首先解释为一个fun 前面的expr是a.foo

不懂的时候用自己的gdb调试一下就直到大致结构了

开工：注意自下而上写

semantic有一个功能就是visit访问寻访信息 有些信息是build 的时候就建立的，有的则需要通过这个迭代器依此访问（我的一些感想），所以要注意好信息的传递，不要传递少了，多的也没必要传递，因为很多信息是建树的时候就完成了，没必要重复工作，不会偷懒的程序员不是好的程序员

scope基本

我设计的typenode map是为了记录出现过的类型（用set也行）实现的方法很多，放开写就行

```
public class GlobalScope implements Scope{}
```

```
private Map<String, VarSymbol> varSymbolTable;
private Map<String, FuncSymbol> funcSymbolTable;
private Map<String, ClassSymbol> classSymbolTable;
private NullType nullType; //maybe need to be modified
```

```
public class GlobalScope implements Scope{
    private Map<String, VarSymbol> varSymbolTable;
    private Map<String, FuncSymbol> funcSymbolTable;
    private Map<String, ClassSymbol> classSymbolTable;
    private NullType nullType; //maybe need to be modified
```

这是一个好的架构

从下而上开工

计划把所有的expr类型都存在expr虚类的那个标记里面

问题？到底是从上往下写还是从下往上写

过掉自己造的数据点

最后完成

开始visit program的时候 先symbolcollector 然后进行每一个class的信息收集 然后收集每一个function的信息

The screenshot shows a web browser window with a tab bar at the top containing various links like compiler, C, cc, A, C, c, yxy, c, ca, hz, c, hbh, java, riscv, y, G, a, D, x, and a GitHub link. The main content area is titled "Details for Record #96". It contains three sections: "Commit Information", "Build Message", and "Judgement Protocol".

Commit Information:

```
commit 0c1de0ba40a57a387e6185df3687f58d6a13ca7d
Author: yichuan<7376632+yichuan520030910320@users.noreply.github.com>
Date:   Sun Oct 24 23:35:09 2021 +0800

obj
```

Build Message:

```
=git stdout=
=git stderr=
=stdout=
```

Judgement Protocol:

#	Phase	Test case	Verdict	Compiling	Execution Cycles	Judge Time
0	O-Build		1			100.0%
1	1-Semantic		190			100.00%

bingo

四.ir

借鉴

- hz

irbuilder的时候初始化了module 在module构造函数的时候然后把内建函数都塞进去

用栈来维护break continue的信息

然后通过类似symbolcollector 来一个一个查找是不是class 第一次访问收集名字...?

第二次访问全部class....?收集内部的信息....?(vardef 有无参数的constructor 一些function (同样要收集里面的函数参数的信息 来在function里面添加参数信息 由于支持前向引用)) (这些的目的都是往module里面加上一些collector)

第三次全部访问function 来收集function 的信息 这边把类型都弄成i32 i1这些 它的做法是把function的类型里面记录了所有的参数类型(并且在这个function里面记录了所有para类型和名字 (相当于变量声明...?))和返回值类型 (一个returntype 一个arratiss记录参数类型) 初始化的时候 irfunction记录了一个entry 和一个return

然后vardef 访问irbuilder的vardef节点 **init 函数放入**: 只有赋值的变量才能放入 单纯的声明不用放入

然后依次访问class 和function? 可以全部一起访问嘛?

accept去astnode的访问者模式 由于irbuilder继承了接口 可以直接访问irbuilder(递归访问直到最小单元)

函数: 首先先开一个basic block收集参数 然后一个一个statement开始访问

细节statement:

for 每一个init condition step都需要新开一个bb

- Irh:

所有的单元都继承了一个irobj 里面包含它的user value作为基类

basicblock里面套一个链表或者用instruction 的头尾来实现

Basic Components

Module -- Global Define && Function -- BasicBlock -- IRInstruction

在数值define的时候加上init块内部! !

开始printer的时候先

日志

ir是为了平滑从树形到riscv的线性结构

一般不会选择树形的ir

ssa静态单变量赋值形式

尽可能的拿掉ast的特征 达到线性的目的

tac 三地址码

while for的抽象都可以 用if来表示

clang -llvm

ir 无限寄存器

11. 30学习llvm

```
clang -S -emit-llvm test.c
```

生成中间代码

```
clang -S -emit-llvm -O3 test.c
```

O3优化

```
llc test.ll
```

生成汇编代码

然后用操作系统自带的汇编器和链接器生成可执行文件

这是一个基于LLVM的编译器

```
.c --frontend--> AST --frontend--> LLVM IR --LLVM opt--> LLVM IR --LLVM llc--> .s  
Assembly --OS Assembler--> .o --OS Linker--> executable
```

- clang

```
# 生成可执行文件  
$ clang main.c -o main  
# 查看编译的过程  
$ clang -ccc-print-phases main.c  
  
# 生成 tokens  
$ clang -E -xclang -dump-tokens main.c  
# 生成语法树  
$ clang -fsyntax-only -xclang -ast-dump main.c  
# 生成 LLVM IR (不开优化)  
$ clang -S -emit-llvm main.c -o main.ll -O0  
  
# 生成汇编 (在本实验中用处不大)  
$ clang -S main.c -o main.s  
# 生成目标文件 (在本实验中用处不大)  
$ clang -c main.c -o main.o
```

clang -S -emit-llvm main.c 可以生成.ll文件

```
# 1. 生成 main.c 对应的 .ll 格式的文件  
$ clang -S -emit-llvm main.c -o main.ll -O0  
  
# 2. 用 lli 解释执行生成的 .ll 文件  
$ lli main.ll  
  
$ clang main.ll -o code
```

12.3

学LLVM

我们的IR可以用lli来查看结果

12.4

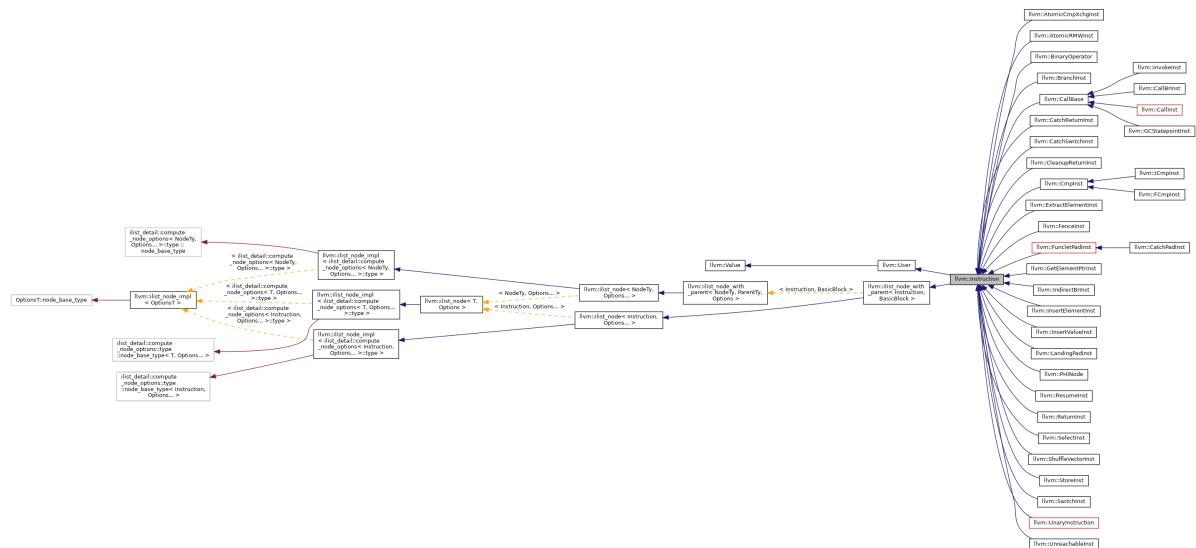
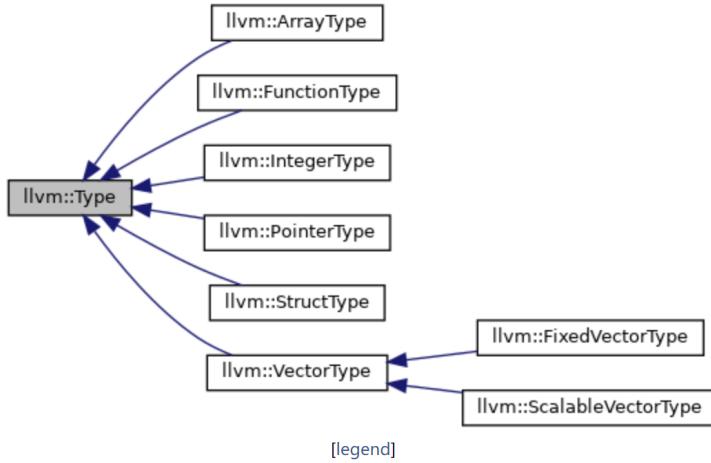
还在学 其实可以开始写了时间紧迫

12.5

开始写ir

bool 存储与load的时候长度变化

之后可以加上use链用于优化...?也可以现在加



12.6

```
const Use &getOperandUse(unsigned i) const {
    assert(i < NumUserOperands && "getOperanduse() out of range!");
    return getOperandList()[i];
}
Use &getOperandUse(unsigned i) {
    assert(i < NumUserOperands && "getOperanduse() out of range!");
    return getOperandList()[i];
}
```

赵一龙说要allign 4

经过测试struct 可以后面直接跟i32 i8 不用像clang那样子采用复杂的形式

//todo加上use value链

12.7开始设计bb function module

12.8

那些instruction里面用到的操作数 暂时不用处理 因为这些都已经存储在具体节点内部 br也不用存储bb
也存储在内部节点里面了

begin to write irbuilder

bool 采取mallocate 1的写法算了

争取这周内实现能跑的llvm!!!

所有的expr里面都加上operand 然后lvalueoperand可以后面加

一个函数的entrance 块进来如果有return值 我都需要先给return allocate i32

所有的returnblock都只有两行 load 和 ret

开始实现它吧

然后弄完之后去把print 搞出来

先不用管那么多细节 大致实现自己清楚就好

先写着后面更改起来也是挺容易的

每一个expr结束都需要把自己的operand设置为一个register

return 块在构造的时候先出现 但是在所有函数的stat都访问完之后才放入linkedlist 这样能够保持
linkedlist的顺序性

//todo

emmm

1.加上内建函数 (部分内建函数还没有实现) emmm两边都存储一下好了emmm这样子可以保证所有
外部的声明都打印在一起2.研究irprinter的写法 (**特别是计数器问题**) 尝试今天能够输出

思考指针的层级问题是如何处理的

编译器设计 (书籍)

先实现部分的内建函数 (print 相关)

计数问题的解决方法是在currentfunction里面维护一个symboltable()如果存在的话就改变名字 (后面数
字增加) 不对这个是维护bb的名字问题

```
21     arrayList = symbolTable.get(name);    name: "call"    symbolTable: size
22     label = arrayList.size();
23 } else {
24     arrayList = new ArrayList<>();
25     symbolTable.put(name, arrayList);
26     label = 0;
27 }
28
29 if (object instanceof BasicBlock = false )
30     ((BasicBlock) object).setName(name + "." + label);
31 else if (object instanceof Parameter = false )
32     ((Parameter) object).setName(name + "." + label);
33 else if (object instanceof Register = true )
34     ((Register) object).setName(name + "." + label);
35 else
```

重新命名的方法

12.9

- 处理重命名问题...? (用function里面的一个register map 一个bb map就可以解决) 同时处理 parament 作为register的情况 emmm在构造函数的时候可以解决这个问题 (register 加入 function里面的reg map里面) //todo emmm或者用cnt解决

使用renaming map处理 需要在new reg 和new bb的时候手动调用

- 处理这个东西...?思考

- %call.0 = call i32 @_Z5sets1i(i32 %0) 归于call

- 完成println(1) (先完成这个!)
- 完成binary unary operation 提高代码的复用率
- 加上global 以及init块 一个大的问题是 valdecl init函数
- 完成 print ("hello world") 这里要处理get_elementptr 和 string //todo 简单的内部string emmm这边采用简单的写法 直接往print里面传 getelementptr ? 问xt emmm 我的处理时 存一个 string globalvar map
%stringLiteral.0 = getelementptr [12 x i8], [12 x i8]* @.str.0, i32 0, i32 0 后面的i32 0是写死的 emm 第一个i32 0代表是地址的偏移量 在string里面一直一直也是i32 0 一个type 一个type * 然后后面加上偏移量
- 思考指针的层级问题是如何处理的

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @_Z5sets1i(i32 %0) #0 {
    %2 = alloca i32, align 4
    store i32 %0, i32* %2, align 4
    %3 = load i32, i32* %2, align 4
    ret i32 %3
}
```

不同function的寄存器可以使用同一个名字 所以我需要在function里面维护一个使用过register的名字类似的还有 bb

争取这周能开始跑过大部分点（甚至过掉llvm）

本质就是ast 到ir 线性结构的转化 所以有一大堆的转化函数 本质就是线性变成非线性

12.10

函数进来时候如果有数值的定义emmm 然后在entrance块的头部进行空间的分配 init 操作都在当前块

scope 一个id 比如x=10 要先去找给id 分配的空间 emmm 可以在astnode里面记录信息 lrh是 varentity emmm也可以在scope里面记录信息 string到 reg的映射(emmm function里面记录 scope emm还是其他方法 自己新建立维护一个scope) emml临时变量不需要记录在regmap里面 只有valdecl需要考虑....? emmm临时值也没有地址 故不需要考虑 地址到寄存器的映射 load store 的寄存器存储的均是地址

加上前面块的注释

之后可以用思维导图画一幅 结构图java里面的继承关系

emmm 为了输出而加功能 可以分块测试

12.11

关于get element ptr 的理解

1.

```
long *nums = {1, 2, 3};  
long index_first(void) {  
    return nums[0];  
}
```

```
@nums = dso_local global i64* inttoptr (i64 1 to i64*), align 8  
  
define dso_local i64 @index_first() #0{  
    %0 = load i64*, i64** @nums, align 8  
    %arrayidx = getelementptr inbounds i64, i64* %0, i64 0  
    %1 = load i64, i64* %arrayidx, align 8  
    ret i64 %1  
}
```

2.

```
long nums[3][3] = { {1, 2, 3}, {2, 3, 4}, {3, 4, 5} };  
long i;  
long index_i2(void) {  
    return nums[i][i];  
}
```

```

@nums = dso_local local_unnamed_addr global [3 x [3 x i64]] [[3 x i64] [i64 1,
i64 2, i64 3], [3 x i64] [i64 2, i64 3, i64 4], [3 x i64] [i64 3, i64 4, i64 5]], align 16
@i = common dso_local local_unnamed_addr global i64 0, align 8

define dso_local i64 @index_i2() local_unnamed_addr #0 {
    %0 = load i64, i64* @i, align 8
    %arrayidx1 = getelementptr inbounds [3 x [3 x i64]], [3 x [3 x i64]]* @nums,
i64 0, i64 %0, i64 %0
    %1 = load i64, i64* %arrayidx1, align 8
    ret i64 %1
}

```

注意到depoint 总是解引用一层 emmm后面描述的树不同维数的偏移量 emmm

new 作为一个loop出现



```

int main() {
int i=0;
{
int i=9;
printlnInt(i);
}
printlnInt(i);
return 0;
}

```

finish the above

```

#include "stdio.h"
int k=9;
int p=k;
int main() {
int i=1; printf("%d",i);
return 0;
}

```

c库的头文件

make the following init

```

@k = dso_local global i32 4, align 4
@p = dso_local global i32 0, align 4
@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_main.mx, i8* null }]

```

```

; Function Attrs: noinline norecurse optnone uwtable
define i32 @main() #1 {
    %1 = alloca i32, align 4
    store i32 0, i32* %1, align 4
    %2 = load i32, i32* @p, align 4
    %3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 %2)
    ret i32 0
}

declare dso_local i32 @printf(i8*, ...) #2

define void @_GLOBAL__sub_I_main.mx() {
    %1 = load i32, i32* @k, align 4
    store i32 %1, i32* @p, align 4
    ret void
}

```

```

@k = dso_local global i32 9, align 4
@p = dso_local global i32 0, align 4
@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32 65535, void ()* @_GLOBAL__sub_I_main.mx, i8* null }]

; Function Attrs: noinline norecurse optnone uwtable
define dso_local i32 @main() #1 {
    %1 = alloca i32, align 4
    %2 = alloca i32, align 4
    store i32 0, i32* %1, align 4
    store i32 1, i32* %2, align 4
    %3 = load i32, i32* %2, align 4
    %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 %3)
    ret i32 0
}

declare dso_local i32 @printf(i8*, ...) #2

; Function Attrs: noinline uwtable
define void @_GLOBAL__sub_I_main.mx() {
    %1 = load i32, i32* @k, align 4
    store i32 %1, i32* @p, align 4
    ret void
}

```

```

int k=9;
int p=k;
int main() {
    printlnInt(p);
    return 0;
}

```

缓存不要太大一步一步走

还是采用main函数call init 的方法吧 emmm这样子可以便于后续的codegen而尽量减少clang的调用

我不能做到return void很多函数只有一个Block那样 我没有实现它的水平qwqq

我只能跳转到return快

bingo

总结todo

- 控制流 for 循环 if 等变化
- 函数调用
- 处理复杂return情况
- 进入函数的时候要求先给parament allocate空间 然后进行操作 给参数分配空间 结论是需要分配

测试一下bool 增加拓展性...?maybe

数组 (数组长度存在-1位)

单目运算符

12.12

```
void calltest(int i){  
    printlnInt(i);  
}  
int main() {  
    calltest(7);  
    return 0;  
}
```

测试点

用main.c clang只能编译.c文件

```
define dso_local i32 @k(i32 %0) #0 {  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 %0, i32* %3, align 4  
    %4 = load i32, i32* %3, align 4  
    %5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x  
i8]* @.str, i64 0, i64 0), i32 %4)  
    %6 = load i32, i32* %2, align 4  
    ret i32 %6  
}
```

%3是存储内存地址的一个寄存器 %0是一个具体的数值 先把para记录下来 记录string+"para"和寄存器的映射 然后string 和一个新的寄存器建立映射

注意我的para 寄存器名字都加xxx_para

1213

then 肯定存在 else 不一定存在

todo 我的现在这个不行qwqq

```
int foo(int k){  
    printlnInt(k);}  
int main() {  
    int c=9;  
    c=c+8;  
    int b=7;  
    printlnInt(b);  
    printlnInt(c);  
  
    return 0;  
}
```

处理 重复赋值 会出问题

完成赋值和一元表达式

id出问题 emmm idexpr

关于右移计算 mx采用的是算数右移 由于mx用的都是有符号数

12.14

- return 现在的问题是有两个branch指令 解决惹
- for while
- array
- 短路求值

短路求值采用 创建新的block的方法 不用按位or或者按位与 直接用icmp实现

```
if(a==1) off  
else b70
```

current b
itter

Condition

- 5 -

$$\alpha = -1$$

100

Demand

余秋雨

11

cont.

~~✓✓✓✓~~

rt C. soon
it soon
is operating on
the

1

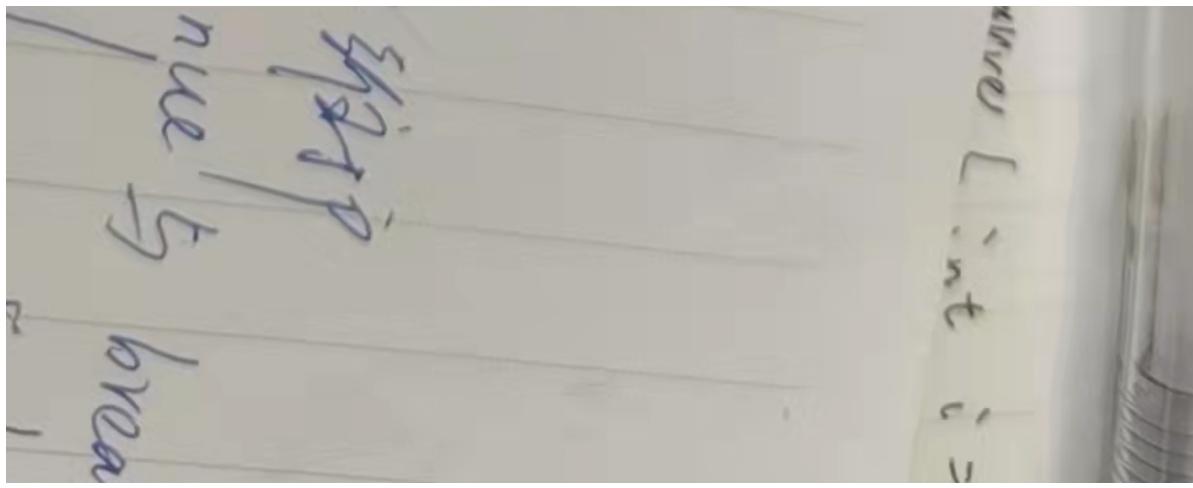
short circuit

conductance

A hand-drawn diagram of a neuron on lined paper. The neuron has a central soma with three processes: an axon extending to the right, a dendrite branching downwards, and a shorter process extending to the left. Red arrows point from the text labels to their corresponding parts of the neuron.

Labels in red ink:

- short C → short process
- long br. → long branching dendrite
- cond. → conductor (likely referring to the axon)



在irbuilder阶段始终需要访问irlhs但事实上运行中不一定会跳转到branch的块

store bool pointer--->i1

emmm 先在stack中alloca 在两个块都可以store这边的值

continue break (用一个stack来维护 就挺简单的emmm但不太好想)

```
/*
Test Package: Codegen
Author: Admin
Time: 2020-02-02
Input:
==== input ====
==== end ====
Output:
==== output ====
==== end ====
ExitCode: 10
InstLimit: -1
Origin Package: Codegen Pretest-577
*/
```

```
int main() {
    int i;
    for(i=0;i<6;i++){printlnInt(i);
        if(i>4)break;
    }
    return 0;
}
```

//to pass it

string 加法 大小比较 先写 可以跑点

12 字符串:

12.1 字符串对象

字符串对象赋值为null是语法错误。

12.2 字符串的双目运算

+表示字符串拼接

`==`, `!=`比较两个字符串是否完全一致（不是内存地址）

`<`, `>`, `<=`, `>=`用于比较字典序大小

剩余双目运算符都是语法错误，字符串双目运算符要求两边类型相同，不满足则语法错误。

12.3 字符串的内建方法

函数: `int length();`

使用: `<StringIdentifier>.length();`

作用: 返回字符串的长度。

函数: `string substring(int left, int right);`

使用: `<StringIdentifier>.substring(left, right);`

作用: 返回下标为`[left, right)`的子串。

函数: `int parseInt();`

使用: `<StringIdentifier>.parseInt();`

作用: 返回一个整数，这个整数应该是该字符串的最长前缀。如果该字符串没有一个前缀是整数，结果未定义。如果该整数超界，结果也未定义。

函数: `int ord(int pos);`

使用: `<StringIdentifier>.ord(pos);`

作用: 返回字符串中的第`pos`位上的字符的ASCII码。下标从0开始编号。

常量字符串不具有内建方法，使用内建方法的常量字符串未定义。

由于semantic写的过于屎山 我的type判断不能使用`instance of` 而只能用`type.typename`来判断string
注意这边可能导致typetrans 产生问题 可以用`typename`特判一下

todo the next point

```
int A = 1;
int B = 1;
int C = 1;

int main(){
    //while (C < (1 << 29) && C > -(1 << 29)) {
```


char * 和string 都是i8*

- string =//感觉是不会用到的?
- 加上一个print()的 built in 貌似漏了一个

12.15

- 拓展一下for
- 开始array 弄完array (开始! ! ! !)
- class

```
int main() {  
    int []a=new int[5];  
    int i=0;  
    for(i=0;i<5;i++)a[i]=i;  
    for(i=0;i<5;i++)printlnInt(a[i]);  
  
    return 0;  
}
```

to pass it

```
/*  
Test Package: Codegen  
Author: 11' Hang Wu  
Time: 2020-01-25  
Input:  
==== input ====  
==== end ====  
Output:  
==== output ====  
2  
==== end ====  
ExitCode: 0  
InstLimit: -1  
Origin Package: Codegen Pretest-538  
*/
```

```
//int[] a = new int[4];  
int main()  
{  
    int[][] b = new int[4][2];  
    b[2][1]=2;  
    //a=b;  
    //println(tostring(a[2]));  
    println(tostring(b[2][1]));  
  
    return 0;  
}
```

```

/*
Test Package: Codegen
Author: 11' Hang Wu
Time: 2020-01-25
Input:
==== input ====
==== end ====
Output:
==== output ====
2
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-538
*/
//int[] a = new int[4];
int main()
{
    int[] b = new int[4];
    b[2]=2;
    //a=b;
    //println(toString(a[2]));
    println(toString(b[2]));

    return 0;
}

```

to work it

array 作为bianry expr的左值的时候

采用特判的办法 在arrayexp记录额外的数据

我的getelement ptr是每次都解引用一层的emmm 所以后面的参数列表都只有一个 采用多行多次的方法 而不是一步到位

12 16完成多维数组和class

todo

```

/*
Test Package: Codegen
Author: 11' Hang Wu
Time: 2020-01-25
Input:
==== input ====
==== end ====
Output:
==== output ====
2
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-538
*/

```

```
//int[] a = new int[4];
int main()
{
    int[][] b = new int[4][2];
    b[2][1]=2;
    //a=b;
    //println(toString(a[2]));
    println(tostring(b[2][1]));

    return 0;
}
```

class

12.17

```
/*
Test Package: Codegen
Author: 10' Huan Yang
Time: 2020-01-24
Input:
--- input ---
102
--- end ---
Output:
--- output ---
68
--- end ---
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-523
*/
bool check(int a, int N) {
    return ((a < N) && (a >= 0));
}

int main() {
    int N;
    int head;
    int startx;
    int starty;
    int targetx;
    int targety;
    int tail;
    int ok;
    int now;
    int x;
    int y;
    int[] xlist;
    int[] ylist;
    int[][] step;
    int i;
    int j;

    N = getInt();
```

```

head =0;
tail = 0;
startx = 0;
starty = 0;
targetx = N-1;
targety = N-1;
x = 0;
y = 0;
now = 0;
ok = 0;
xlist = new int[N * N];
for (i = 0; i < N * N; i ++ )
    xlist[i] = 0;
ylist = new int[N * N];
for (i = 0; i < N * N; i ++ )
    ylist[i] = 0;
step = new int[N][][];
for (i = 0; i < N; i ++ ) {
    step[i] = new int[N];
    for (j = 0; j < N; j ++ )
        step[i][j] = -1;
}
xlist[0] = startx;
ylist[0] = starty;
step[startx][starty] = 0;
while (head <= tail)
{
    now = step[xlist[head]][ylist[head]];
    x = xlist[head] - 1;
    y = ylist[head] - 2;
    if (check(x, N) && check(y, N) && step[x][y] == -1)
    {
        tail = tail + 1;
        xlist[tail] = x;
        ylist[tail] = y;
        step[x][y] = now + 1;
        if (x == targetx && y == targety) ok = 1;
    }
    x = xlist[head] - 1;
    y = ylist[head] + 2;
    if (check(x, N) && check(y, N) && step[x][y] == -1)
    {
        tail = tail + 1;
        xlist[tail] = x;
        ylist[tail] = y;
        step[x][y] = now + 1;
        if (x == targetx && y == targety) ok = 1;
    }
    x = xlist[head] + 1;
    y = ylist[head] - 2;
    if (check(x, N) && check(y, N) && step[x][y] == -1)
    {
        tail = tail + 1;
        xlist[tail] = x;
        ylist[tail] = y;
        step[x][y] = now + 1;
        if (x == targetx && y == targety) ok = 1;
    }
}

```

```

x = xlist[head] + 1;
y = ylist[head] + 2;
if (check(x, N) && check(y, N) && step[x][y] == -1)
{
    tail = tail + 1;
    xlist[tail] = x;
    ylist[tail] = y;
    step[x][y] = now + 1;
    if (x == targetx && y == targety) ok = 1;
}
x = xlist[head] - 2;
y = ylist[head] - 1;
if (check(x, N) && check(y, N) && step[x][y] == -1)
{
    tail = tail + 1;
    xlist[tail] = x;
    ylist[tail] = y;
    step[x][y] = now + 1;
    if (x == targetx && y == targety) ok = 1;
}
x = xlist[head] - 2;
y = ylist[head] + 1;
if (check(x, N) && check(y, N) && step[x][y] == -1)
{
    tail = tail + 1;
    xlist[tail] = x;
    ylist[tail] = y;
    step[x][y] = now + 1;
    if (x == targetx && y == targety) ok = 1;
}
x = xlist[head] + 2;
y = ylist[head] - 1;
if (check(x, N) && check(y, N) && step[x][y] == -1)
{
    tail = tail + 1;
    xlist[tail] = x;
    ylist[tail] = y;
    step[x][y] = now + 1;
    if (x == targetx && y == targety) ok = 1;
}
x = xlist[head] + 2;
y = ylist[head] + 1;
if (check(x, N) && check(y, N) && step[x][y] == -1)
{
    tail = tail + 1;
    xlist[tail] = x;
    ylist[tail] = y;
    step[x][y] = now + 1;
    if (x == targetx && y == targety) ok = 1;
}
if (ok == 1) break;
head = head + 1;
}
if (ok == 1) println(toString(step[targetx][targety]));
else print("no solution!\n");
return 0;
}

```

```

/*
Test Package: Sema_Local_Preview
Test Target: String
Author: 15' Caomeng Yao
Time: 2019-10-20
Verdict: Success
Origin Package: Semantic Extended
*/

string[] str_arr = null;

int main() {
    int la = getInt();
    str_arr = new string[la];

    int i;
    int cnt = 0;
    for (i = 0; i < la; i++) {
        str_arr[i] = getString();
        cnt = cnt + str_arr[i].length();
    }

    string str = "";
    int sum = 0;
    for (i = 0; i < la; ++i){
        str = str + str_arr[i].substring(0, str_arr[i].length() - 1);
        sum = sum + str_arr[i].ord(0);
    }
    println(str);
    print(toString(sum));
    if (cnt == str.length()) return 0;

    else return 1;
}

```

t1 t3 短路求值烂了qwqqq

class!!

12.18

class完成

在getelementptr上面有一个写死的1 第一位

```

class m{
    int k;
    int m;
};

int main(){
    m M=new m;
    M.k=9;
    printlnInt(M.k);
    return 0;
}

/*

```

```
Test Package: Codegen
Author: Admin
Time: 2020-02-02
Input:
==== input ====
==== end ====
Output:
==== output ====
==== end ====
ExitCode: 70
InstLimit: -1
Origin Package: Codegen Pretest-574
*/
class C2 {
    int x;
    int y;
    int z;
};

int main() {
    C2 obj = new C2;
    obj.x = 10;
    obj.y = 20;
    obj.z = 40;
    return obj.x + obj.y + obj.z;
}
```

```
class m{
int k;
int m;
MMM o;
m(){
o=new MMM;

}
};

class MMM{
int g=8;
MMM(){
g=6;
}

};

int main(){
m M=new m;
M.k=9;
printlnInt(M.k);
printlnInt(M.o.g);

return 0;
}
```

如果是struct type则需要转化为指针类型

llvm是强类型语言 alloca出来指针还是数据都是32位的emmm class*也是这样的

- 加上align qwqqqq
- 数组的size
- wsl java环境配置
- return在前 有些细节还没处理

12.19

class llvm 的gep 位置是该元素在class中的第几个元素

```
%class.C2 = type { i32, i32, i32 } 这个应该理解成一个type而不是一个寄存器
```

-

```
• /*
Test Package: Codegen
Author: Admin
Time: 2020-02-02
Input:
==== input ====
==== end ====
Output:
==== output ====
==== end ====
ExitCode: 2
InstLimit: -1
Origin Package: Codegen Pretest-587-Modifiy
*/
int main() {
    string s = "hahaha";
    return s.substring(2, 4).length();
}
```

- class 内函数 加上this指针的传入 再加上functioncall 注意function里面的参数全部存储在 functiontype里面 自己的设计

```
class K{
    int m;
    void test(){
        println(m);
    }
};

int main(){
    K tmp=new K;
    tmp.m=9;
    tmp.test();
    return 0;
}
```

```

    public void visit(IdExp_ASTnode it) {
        //naive type but it work now don't find bug
        BaseOperand id_reg = current_ir_scope.find_id_to_reg(it.index);
        //single val decl before
        if (id_reg != null) {
            Register load_reg = new Register(type_trans.asttype_to_irtype(it.type), it.index);
            current_function.renaming.add(load_reg);
            current_basicblock.instruction_add(new LoadInstruction(current_basicblock, load_reg, id_reg));
            it.ir_operand = load_reg;
        } else {
            //in class where don't have value decl before so we load the data from the heap using gep
        }
        //todo
    }
}

```

我的设计是刚开始class里面最先遍历到的id gep一下之后都可以用这个公用的值 目前看来没问题

class内部函数调用 emmm this指针使用

```

/*
Test Package: Codegen
Author: 14' Rongyu You
Time: 2020-02-03
Input:
==== input ====
==== end ====
Output:
==== output ====
vector x: ( 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 )
excited!
vector y: ( 9, 8, 7, 817, 5, 4, 3, 2, 1, 0 )
x + y: ( 18, 16, 14, 823, 10, 8, 6, 4, 2, 0 )
x * y: 0
(1 << 3) * y: ( 72, 64, 56, 6536, 40, 32, 24, 16, 8, 0 )
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-900
*/
//
// Naive vector class for Mx*.
// without any guarantee for robustness.
//
class vector{
    int[] data;
    void init(int[] vec){
        // init the vector from an array
        if (vec == null) return;
        data = new int[vec.size()];
        int i;
        for (i = 0; i < vec.size(); ++i)
        {
            data[i] = vec[i];
        }
    }

    int getDim(){
        if (data == null) return 0;
        return data.size();
    }
}

```

```

int dot(vector rhs){
    int i = 0;
    int result = 0;
    while(i < getDim()){
        //result = data[i] * rhs[i];
        result = data[i] * rhs.data[i];
        ++i;
    }
    return result;
}

vector scalarInPlaceMultiply(int c){
    if (data == null) return null;
    int i;
    for (i = 0; i < getDim(); ++i) {
        this.data[i] = c * this.data[i];
    }
    return this;
}

vector add(vector rhs){
    if (getDim() != rhs.getDim() || getDim() == 0)
        return null;
    vector temp = new vector;
    int i;
    temp.data = new int[getDim()];
    for (i = 0; i < getDim(); ++i){
        temp.data[i] = data[i] + rhs.data[i];
    }
    return temp;
}

bool set(int idx, int value){
    if (getDim() < idx) return false;
    data[idx] = value;
    return true;
}

string tostring(){
    string temp = "(" ;
    if (getDim() > 0) {
        temp = temp + toString(data[0]);
    }
    int i;
    for (i = 1; i < getDim(); ++i) {
        temp = temp + ", " + toString(data[i]);
    }
    temp = temp + " )";
    return temp;
}

bool copy(vector rhs){
    if (rhs == null) return false;
    if (rhs.getDim() == 0) {
        data = null;
    } else {
        data = new int[rhs.getDim()];
        int i;

```

```

        for (i = 0; i < getDim(); ++i) {
            data[i] = rhs.data[i];
        }
    }
    return true;
}
};

int main(){
    vector x = new vector;
    int[] a = new int[10];
    int i;
    for (i = 0; i < 10; ++i){
        a[i] = 9 - i;
    }
    x.init(a);
    print("vector x: ");
    println(x.tostring());

    vector y = new vector;
    y.copy(x);
    if (y.set(3, 817)){
        println("excited!");
    }
    print("vector y: ");
    println(y.tostring());
    print("x + y: ");
    println((x.add(y)).tostring());
    print("x * y: ");
    println(tostring(x.dot(y)));
    print("(1 << 3) * y: ");
    println(y.scalarInPlaceMultiply(1 << 3).tostring());
    return 0;
}

```

return this && string equal

我使用 module_in_irbuilder.Module_Struct_Map.get(current_class_detail.classname) 来找到当前 class 的类型

to pass constructor

```
/*
Test Package: Codegen
Author: 14' Xingyuan Sun
Time: 2020-02-03
Input:
==== input ====
==== end ====
Output:
==== output ====
(0, 0, 0)
28716325
7421636
9980404
38464544
1854392
```

```
(7616, 1666188, -1232986)
(-508, 4119, 3390)
(562, 1584, 2144)
(-920, 768, -524)
(612, -469, -630)
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-901
*/
class point {
    int x;
    int y;
    int z;
    point() {
        x = 0;
        y = 0;
        z = 0;
    }
    void set(int a_x, int a_y, int a_z){
        x = a_x;
        y = a_y;
        z = a_z;
    }
    int sqrLen(){
        return x * x + y * y + z * z;
    }
    int sqrDis(point other) {
        return (x - other.x) * (x - other.x) + (y - other.y) * (y - other.y) +
(z - other.z) * (z - other.z);
    }
    int dot(point other) {
        return x * other.x + y * other.y + z * other.z;
    }
    point cross(point other) {
        point retval = new point;
        retval.set(y * other.z - z * other.y, z * other.x - x * other.z, x *
other.y - y * other.x);
        return retval;
    }
    point add(point other) {
        x = x + other.x;
        y = y + other.y;
        z = z + other.z;
        return this;
    }
    point sub(point other) {
        x = x - other.x;
        y = y - other.y;
        z = z - other.z;
        return this;
    }
    void printPoint() {
        println("(" + toString(x) + ", " + toString(y) + ", " + toString(z) +
")");
    }
};
```

```

int main() {
    point a = new point;
    point b = new point;
    point c = new point;
    point d = new point;
    a.printPoint();
    a.set(849, -463, 480);
    b.set(-208, 585, -150);
    c.set(360, -670, -742);
    d.set(-29, -591, -960);
    a.add(b);
    b.add(c);
    d.add(c);
    c.sub(a);
    b.sub(d);
    d.sub(c);
    c.add(b);
    a.add(b);
    b.add(b);
    c.add(c);
    a.sub(d);
    a.add(b);
    b.sub(c);
    println(toString(a.sqrLen()));
    println(toString(b.sqrLen()));
    println(toString(b.sqrDis(c)));
    println(toString(d.sqrDis(a)));
    println(toString(c.dot(a)));
    b.cross(d).printPoint();
    a.printPoint();
    b.printPoint();
    c.printPoint();
    d.printPoint();
    return 0;
}

```

有构造函数的话就需要在new A 的时候显示调用构造函数

带参数的构造函数是未定义行为

//todo add constructor

```

/*
Test Package: Codegen
Author: 14' Xingyuan Sun
Time: 2020-02-03
Input:
==== input ====
==== end ====
Output:
==== output ====
(0, 0, 0)
28716325
7421636
9980404
38464544
1854392

```

```

(7616, 1666188, -1232986)
(-508, 4119, 3390)
(562, 1584, 2144)
(-920, 768, -524)
(612, -469, -630)
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-901
*/
class point {
    int x;
    int y;
    int z;
    point() {
        x = 1;
        y = 0;
        z = 0;
    }
    void printPoint() {
        println("(" + toString(x) + ", " + toString(y) + ", " + toString(z) +
")");
    }
}
int main() {
    point a = new point;
    a.printPoint();

    return 0;
}

```

//todo

```

step = new int[N][];
for (i = 0; i < N; i++) {
    step[i] = new int[N];
    for (j = 0; j < N; j++)
        step[i][j] = -1;
}

for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++)
        printlnInt(step[i][j]);
}

```

fuckkkkk! break at this pointer!!

```

/*
Test Package: Optim
Author: zhekai zhang, 15
Input:
==== input ====
1 acm2015
2 ABC64A57029F21F165A96BDB59F0351C7C7D1769

```

```
0

==== end ====
Output:
==== output ====
5B38674EB4BD02CEC1D41C8DE3CC14A9872A2656
ACM

==== end ====
ExitCode: 0
InstLimit: -1
*/
//Compute and crack SHA-1
//by zzk

int hex2int(string x)
{
    int i;
    int result = 0;
    for(i=0;i<x.length();i++)
    {
        int digit = x.ord(i);
        if(digit >= 48 && digit <= 57)
            result = result * 16 + digit - 48;
        else if(digit >= 65 && digit <= 70)
            result = result * 16 + digit - 65 + 10;
        else if(digit >= 97 && digit <= 102)
            result = result * 16 + digit - 97 + 10;
        else
            return 0;
    }
    return result;
}

string asciiTable = " !\"#$%&'()*+,-./0123456789:;=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\`]^_`abcdefghijklmnopqrstuvwxyz{|}~";
string int2chr(int x)
{
    if(x >= 32 && x <= 126)
        return asciiTable.substring(x-32, x-31);
    return "";
}
string toStringHex(int x)
{
    string ret = "";
    int i;
    for(i=28;i>=0;i=i-4)
    {
        int digit = (x >> i) & 15;
        if(digit < 10)
            ret = ret + int2chr(48+digit);
        else
            ret = ret + int2chr(65+digit-10);
    }
    return ret;
}
int rotate_left(int x, int shift)
```

```

{
    if(shift == 1)
        return ((x & 2147483647) << 1) | ((x >> 31) & 1);
    if(shift == 31)
        return ((x & 1) << 31) | ((x >> 1) & 2147483647);
    return ((x & ((1 << (32-shift)) - 1)) << shift) | ((x >> (32-shift)) & ((1 << shift) - 1));
}
int add(int x, int y) //to avoid possible undefined behaviour when overflow
{
    int low = (x & 65535) + (y & 65535);
    int high = (((x >> 16) & 65535) + ((y >> 16) & 65535) + (low >> 16)) &
65535;
    return (high << 16) | (low & 65535);
}
int lohi(int lo, int hi)
{
    return lo | (hi << 16);
}

int MAXCHUNK = 100;
int MAXLENGTH = (MAXCHUNK-1) * 64 - 16;
int[][] chunks = new int[MAXCHUNK][80];
int[] inputBuffer = new int[MAXLENGTH];
int[] outputBuffer = new int[5];
int[] sha1(int[] input, int length)
{
    int nchunk = (length + 64 - 56) / 64 + 1;
    if(nchunk > MAXCHUNK)
    {
        println("nchunk > MAXCHUNK!");
        return null;
    }
    int i;
    int j;
    for(i=0;i<nchunk;i++)
        for(j=0;j<80;j++)
            chunks[i][j] = 0;
    for(i=0;i<length;i++)
        chunks[i/64][i%64/4] = chunks[i/64][i%64/4] | (input[i] << ((3-i%4)*8));
    chunks[i/64][i%64/4] = chunks[i/64][i%64/4] | (128 << ((3-i%4)*8));
    chunks[nchunk-1][15] = length << 3;
    chunks[nchunk-1][14] = (length >> 29) & 7;

    int h0 = 1732584193; //0x67452301
    int h1 = lohi(43913, 61389); //0xEFCDAB89
    int h2 = lohi(56574, 39098); //0x98BADCCE
    int h3 = 271733878; //0x10325476
    int h4 = lohi(57840, 50130); //0xC3D2E1F0
    for(i=0;i<nchunk;i++)
    {
        for(j=16;j<80;j++)
            chunks[i][j] = rotate_left(chunks[i][j-3] ^ chunks[i][j-8] ^
chunks[i][j-14] ^ chunks[i][j-16], 1);

        int a = h0;
        int b = h1;
        int c = h2;

```

```

        int d = h3;
        int e = h4;
        for(j=0;j<80;j++)
        {
            int f;
            int k;
            if(j<20)
            {
                f = (b & c) | ((~b) & d);
                k = 1518500249; //0x5A827999
            }
            else if(j<40)
            {
                f = b ^ c ^ d;
                k = 1859775393; //0x6ED9EBA1
            }
            else if(j<60)
            {
                f = (b & c) | (b & d) | (c & d);
                k = lohi(48348, 36635); //0x8F1BBCDC
            }
            else
            {
                f = b ^ c ^ d;
                k = lohi(49622, 51810); //0xCA62C1D6
            }
            int temp = add(add(add(rotate_left(a, 5), e), add(f, k)), chunks[i]
[j]);
            e = d;
            d = c;
            c = rotate_left(b, 30);
            b = a;
            a = temp;
        }
        h0 = add(h0, a);
        h1 = add(h1, b);
        h2 = add(h2, c);
        h3 = add(h3, d);
        h4 = add(h4, e);
    }
    outputBuffer[0] = h0;
    outputBuffer[1] = h1;
    outputBuffer[2] = h2;
    outputBuffer[3] = h3;
    outputBuffer[4] = h4;
    return outputBuffer;
}

void computeSHA1(string input)
{
    int i;
    for(i=0; i<input.length(); i++)
        inputBuffer[i] = input.ord(i);
    int[] result = sha1(inputBuffer, input.length());
    for(i=0; i<result.size(); i++)
        print(toStringHex(result[i]));
    println("");
}

```

```

int nextLetter(int now)
{
    if(now == 122) //'z'
        return -1;
    if(now == 90) //'Z'
        return 97; //'a'
    if(now == 57) //'9'
        return 65;
    return now + 1;
}

bool nextText(int[] now, int length)
{
    int i;
    for(i=length-1; i>=0; i--)
    {
        now[i] = nextLetter(now[i]);
        if(now[i] == -1)
            now[i] = 48; //'0'
        else
            return true;
    }
    return false;
}

bool array_equal(int[] a, int[] b)
{
    if(a.size() != b.size())
        return false;
    int i;
    for(i=0; i<a.size(); i++)
        if(a[i] != b[i])
            return false;
    return true;
}

void crackSHA1(string input)
{
    int[] target = new int[5];
    if(input.length() != 40)
    {
        println("Invalid input");
        return;
    }
    int i;
    for(i=0;i<5;i++)
        target[i] = 0;
    for(i=0;i<40;i=i+4)
        target[i/8] = target[i/8] | (hex2int(input.substring(i, i+4)) << (1 - (i / 4) % 2) * 16);

    int MAXDIGIT = 4;
    int digit;
    for(digit=1; digit <= MAXDIGIT; digit++)
    {
        for(i=0;i<digit;i++)
            inputBuffer[i] = 48;

```

```

        while(true)
    {
        int[] out = sha1(inputBuffer, digit);
        if(array_equal(out, target))
        {
            for(i=0;i<digit;i++)
                print(int2chr(inputBuffer[i]));
            println("");
            return;
        }
        if(!nextText(inputBuffer, digit))
            break;
    }
}

int main()
{
    int op;
    string input;
    while(true)
    {
        op = getInt();
        if(op == 0)
            break;
        if(op == 1)
        {
            input = getString();
            computeSHA1(input);
        }
        else if(op == 2)
        {
            input = getString();
            crackSHA1(input);
        }
    }
    return 0;
}

```

to do add internal function emmm such as length ord and so on

12.3 字符串的内建方法

函数: `int length();`

使用: `<StringIdentifier>.length();`

作用: 返回字符串的长度。

函数: `string substring(int left, int right);`

使用: `<StringIdentifier>.substring(left, right);`

作用: 返回下标为 `[left, right)` 的子串。

函数: `int parseInt();`

使用: `<StringIdentifier>.parseInt();`

作用: 返回一个整数, 这个整数应该是该字符串的最长前缀。如果该字符串没有一个前缀是整数, 结果未定义。如果该整数超界, 结果也未定义。

函数: `int ord(int pos);`

使用: `<StringIdentifier>.ord(pos);`

作用: 返回字符串中的第`pos`位上的字符的ASCII码。下标从0开始编号。

常量字符串不具有内建方法, 使用内建方法的常量字符串未定义。

emmm i will specially check it

emmm直接两边都用map find 一下就可以了 emmm 感觉很简单

在functioncall里面更改很简单就可以实现

```
/*
Test Package: Optim
Author: Yunwei Ren, 17
Input:
==== input ====
5

==== end ====
Output:
==== output ====
0: 4
1: 4
2: 6
3: 4
4: 4
6: 2
7: 3
8: 2
10: 1
11: 4
13: 2
14: 6
15: 2
18: 4
19: 3
20: 1
21: 5
22: 1
24: 3
25: 3
26: 2
27: 2
```

30: 5
33: 16
35: 2
36: 1
39: 4
40: 1
41: 8
42: 7
43: 2
44: 2
46: 5
47: 1
48: 2
51: 2
55: 5
57: 2
60: 1
63: 2
64: 1
65: 2
66: 2
67: 2
68: 1
69: 1
75: 1
76: 2
77: 2
78: 1
80: 4
81: 5
82: 2
83: 1
84: 2
86: 4
87: 2
89: 5
90: 6
91: 4
92: 6
93: 1
94: 1
97: 5
99: 1
102: 1
105: 1
106: 2
107: 5
108: 2
109: 5
111: 3
112: 7
115: 2
116: 5
117: 2
118: 1
119: 1
120: 3
121: 3

```

122: 8
126: 2
127: 1

==== end ====
ExitCode: 0
InstLimit: -1
*/



class Node {
    Node pnt;
    Node[] children;
    int key;
    int duplicate;
};

Node constructNode(int key, Node pnt, Node lchild, Node rchild) {
    Node node = new Node;
    node.children = new Node[2];
    node.key = key;
    node.duplicate = 1;
    node.pnt = pnt;
    node.children[0] = lchild;
    node.children[1] = rchild;
    return node;
}

Node root = null;

int insertImpl(Node cur, Node pnt, int childId, int key) {
    if (cur == null) {
        cur = constructNode(key, pnt, null, null);
        pnt.children[childId] = cur;
        return 0;
    }
    if (cur.key == key) {
        ++cur.duplicate;
        return 1;
    }
    int id = 0;
    if (cur.key < key)
        id = 1;
    return insertImpl(cur.children[id], cur, id, key);
}

// return 1 if isIn
int insert(int key) {
    if (root != null)
        return insertImpl(root, null, 0 - 1, key);
    root = constructNode(key, null, null, null);
    return 0;
}

Node findLargest(Node cur) {
    if (cur.children[1] == null)
        return cur;
    return findLargest(cur.children[1]);
}

```

```

int eraseImpl(Node cur, Node pnt, int childId, int key) {
    if (cur == null)
        return 0;
    if (cur.key > key)
        return eraseImpl(cur.children[0], cur, 0, key);
    if (cur.key < key)
        return eraseImpl(cur.children[1], cur, 1, key);
    --cur.duplicate;
    if (cur.duplicate > 0)
        return 1;
    // assert(cur.duplicate == 0);
    if (cur.children[0] == null) {
        if (pnt != null)
            pnt.children[childId] = cur.children[1];
        if (cur.children[1] != null)
            cur.children[1].pnt = pnt;
        if (key == root.key)
            root = cur.children[1];
        return 1;
    }
    Node replacement = findLargest(cur.children[0]);
    if (key == root.key)
        root = replacement;
    // assert(replacement.children[1] == null);
    if (replacement.key != cur.children[0].key) {
        replacement.pnt.children[1] = replacement.children[0];
        if (replacement.children[0] != null)
            replacement.children[0].pnt = replacement.pnt;
    }
    if (pnt != null)
        pnt.children[childId] = replacement;
    replacement.pnt = pnt;
    replacement.children[1] = cur.children[1];
    if (cur.children[1] != null)
        cur.children[1].pnt = replacement;
    if (replacement.key != cur.children[0].key) {
        replacement.children[0] = cur.children[0];
        cur.children[0].pnt = replacement;
    }
    return 1;
}

// return 1 if isIn
int erase(int key) {
    if (root == null)
        return 0;
    return eraseImpl(root, null, -1, key);
}

void printTree(Node cur) {
    if (cur == null)
        return;
    printTree(cur.children[0]);
    println(toString(cur.key) + ":" + toString(cur.duplicate));
    printTree(cur.children[1]);
}

```

```

int MAX = 128;
int MaxRandInt = ~(1 << 31);
int seed;

// In mx, we do not have unsigned int. Hence, we only use the least significant
// 31 bits of an integer here.
int randInt31() {
    int x = seed;
    x = x ^ (x << 13);
    x = x & ~(1 << 31);
    x = x ^ (x >> 17);
    x = x ^ (x << 5);
    x = x & ~(1 << 31);
    seed = x;
    return x;
}

// probability = p / PM
int randop(int n) {
    if (randInt31() < n) {
        return 1;
    }
    return 2;
}

void generateOperations(int n, int t) {
    int i;
    for (i = 0; i < t; ++i) {
        int value = randInt31() % MAX;
        if (randop(n) == 1) {
            insert(value);
        } else {
            erase(value);
        }
    }
}

int main() {
    seed = getInt();
    int m = 50000;
    generateOperations(8 * (MaxRandInt / 10), m);
    generateOperations(2 * (MaxRandInt / 10), 2 * m);
    generateOperations(4 * (MaxRandInt / 10), m);
    printTree(root);
    return 0;
}

```

寄

五.codegen

借鉴与学习

- 汇编与riscv基础

在汇编中，一般用分号来作为注释的标志，即"; "

下图显示了六种基本指令格式，分别是：用于寄存器-寄存器操作的 R 类型指令，用于短立即数和访存 load 操作的 I 型指令，用于访存 store 操作的 S 型指令，用于条件跳转操作的 B 型指令，用于长立即数的 U 型指令和用于无条件跳转的 J 型指令。

31	25 24	20 19	15 14	12 11	7 6	0	
	imm[31:12]			rd	0110111	U lui	
	imm[31:12]			rd	0010111	U auipc	
	imm[20:10:1 11 19:12]			rd	1101111	J jal	
	imm[11:0]	rs1	000	rd	1100111	I jalr	
imm[12:10:5]	rs2	rs1	000	imm[4:1 11]	1100011	B beq	
imm[12:10:5]	rs2	rs1	001	imm[4:1 11]	1100011	B bne	
imm[12:10:5]	rs2	rs1	100	imm[4:1 11]	1100011	B blt	
imm[12:10:5]	rs2	rs1	101	imm[4:1 11]	1100011	B bge	
imm[12:10:5]	rs2	rs1	110	imm[4:1 11]	1100011	B bltu	
imm[12:10:5]	rs2	rs1	111	imm[4:1 11]	1100011	B bgeu	
	imm[11:0]	rs1	000	rd	0000011	I lb	
	imm[11:0]	rs1	001	rd	0000011	I lh	
	imm[11:0]	rs1	010	rd	0000011	I lw	
	imm[11:0]	rs1	100	rd	0000011	I lbu	
	imm[11:0]	rs1	101	rd	0000011	I lhu	
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb	
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh	
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw	
	imm[11:0]	rs1	000	rd	0010011	I addi	
	imm[11:0]	rs1	010	rd	0010011	I slti	
	imm[11:0]	rs1	011	rd	0010011	I sltiu	
	imm[11:0]	rs1	100	rd	0010011	I xor	
	imm[11:0]	rs1	110	rd	0010011	I ori	
	imm[11:0]	rs1	111	rd	0010011	I andi	
0000000	shamt	rs1	001	rd	0010011	I slli	
0000000	shamt	rs1	101	rd	0010011	I srli	
0100000	shamt	rs1	101	rd	0010011	I srai	
0000000	rs2	rs1	000	rd	0110011	R add	
0100000	rs2	rs1	000	rd	0110011	R sub	
0000000	rs2	rs1	001	rd	0110011	R sll	
0000000	rs2	rs1	010	rd	0110011	Rslt	
0000000	rs2	rs1	011	rd	0110011	R xor	
0000000	rs2	rs1	100	rd	0110011	R srl	
0000000	rs2	rs1	101	rd	0110011	R sra	
0100000	rs2	rs1	101	rd	0110011	R or	
0000000	rs2	rs1	110	rd	0110011	R and	
0000000	rs2	rs1	111	rd	0110011	I fence	
0000	pred	succ	00000	000	00000	0001111	I fence.i
0000	0000	0000	00000	001	00000	0001111	I ecall
	000000000000		00000	000	00000	1110011	I ebreak
	000000000001		00000	000	00000	1110011	I csrrw
	csr	rs1	001	rd	1110011	I csrrs	
	csr	rs1	010	rd	1110011	I csrrc	
	csr	rs1	011	rd	1110011	I csrrwi	
	csr	zimm	101	rd	1110011	I cssrrsi	
	csr	zimm	110	rd	1110011	I csrrci	
	csr	zimm	111	rd	1110011		

图 2.3: RV32I 带有指令布局, 操作码, 格式类型和名称的操作码映射。(此图基于[Waterman and

31	0
x0 / zero	Hardwired zero
x1 / ra	Return address
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporary
x6 / t1	Temporary
x7 / t2	Temporary
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Function argument, return value
x11 / a1	Function argument, return value
x12 / a2	Function argument
x13 / a3	Function argument
x14 / a4	Function argument
x15 / a5	Function argument
x16 / a6	Function argument
x17 / a7	Function argument
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporary
x29 / t4	Temporary
x30 / t5	Temporary
x31 / t6	Temporary
32	

31	0
pc	
32	

图 2.4: RV32I 的寄存器。第 3 章解释了 RISC-V 调用约定, 各种指针 (sp, gp, tp, fp), 保存寄存器 (s0-s11) 和临时寄存器 (t0-t6) 背后的基本原理 (基于[Waterman and Asanović 2017]的图 2.1 和表

<https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md>

a great insight into asm

关于%lo %hi

%lo(*symbol*)

The low 12 bits of absolute address for *symbol*.

%hi(*symbol*)

The high 20 bits of absolute address for *symbol*. This is usually used with the %lo modifier to represent a 32-bit absolute address.

```

lui      a0, %hi(symbol)    // R_RISCV_HI20
addi    a0, a0, %lo(symbol) // R_RISCV_L012_I

lui      a0, %hi(symbol)    // R_RISCV_HI20
load/store a0, %lo(symbol) (a0) // R_RISCV_L012_I/S

```

- 虚拟寄存器

比如Java在编译的过程中，会在字节码中虚拟出一些与平台无关的寄存器，**执行时根据需要映射到实际的物理寄存器中，如果物理寄存器不够的话，映射到内存中**。其它语言的编译器，在编译的中间步骤也可能用到类似虚拟寄存器的概念。

- Irh

...在codegen之前有一大堆优化都是基于ir的优化 之后就是codegen

先建立一个asm module 传入ir 的module

把ir 的global参数放入asm module 的global参数map

然后同样的把function 也都放进去

最后只访问function 由于class没有什么访问的必要

然后在Instruction visitor里面不断访问各个ir的节点

访问function 有个东西是stackframe 把asm 目前这个function的栈帧设置好（注意不要弄混asm的module function basicblock 和 ir生成的）

然后虚拟寄存器 函数里面先把返回值放入ra寄存器

```
// ----- Save return address -----
```

把参数放入寄存器放不下的放在栈帧里面

然后访问每一个basicblock

codeemitter是输出asm

先是输出一个带有virtual reg的.s

之后是寄存器分配过后真正的.s

```
private Map<VirtualRegister, BaseOffsetAddr> gepAddrMap;
```

虚拟寄存器就是长成这个样子的

- hz

先进行instruction selector

先去访问全局变量

先把每一个函数放入module的map 然后访问其中每一个函数

访问每一个函数 除了builtin

关于p2align 是字节对齐的位数

我刚刚将相同的 C 源代码翻译成 32 位和 64 位 asm 代码，内存总是以相同的方式与相同的指令对齐
.p2align 4,,15 .这是为什么？

最佳答案

.p2align 指令已记录 [here](#) .

第一个表达式是所需的 2 次幂字节对齐。.p2align 4 填充以在 16 字节边界上对齐。.p2align 5 - 32 字节的边界等。

之后进行寄存器分配

[调用约定学习][<https://zhuanlan.zhihu.com/p/111028312>]

具体来说，调用约定一般规定了

- 参数、返回值、返回地址等放置的位置（寄存器、栈或存储器等）
- 如何将调用子过程的准备工作与恢复现场的工作划分到调用者（Caller）与被调用者（Callee）身上

栈指针指向 %sp (x86 %rsp) 存储了一个内存地址 *%sp 存储的是一个 pc 也就是要返回的地址 [见csapp P167](#)

当 Q 返回到 P 的时候 P 的代码可以访问寄存器 %rax (rsi32 的) 的返回值 %a0

左边是目的寄存器

如果函数要使用两个（原来有值的寄存器）寄存器，我们在进入函数之前要首先把这些寄存器放在栈上，等到这个函数结束之后才能够还原 [见csapp P175](#)

关于 codegen 每一个块的名字是函数标号加上块的标号 这样每一个块都是不重名的 这个是 clang 的格式
可以用两个类作为 hashmap key data 这样子可以避免冲突（相比较于 string）

我现在已经把 pointertype 的 byt enum 改成了 8 ir 中间有一个点不能跑过但是用于 codegen 由于 rsc32 是 32 位的

太优美了 lhy 的写法 alloca 在访问的过程中统计

多个 ret 的情况我们采用跳转到最后一个块的方法

日志

1.3

SSA Construction(Mem2Reg in LLVM IR)

Note that LLVM IR is **in SSA form for registers**, but **not for memory**. So there are lots memory access in original LLVM IR. Hence we need to perform a SSA Construction **for memory** so that for all alloca instructions, their corresponding load/store instructions can be removed.

Algorithm

See Chapter 3 of [SSA Book](#) for details.

Step 1: Construct **Dominator Tree** and compute the **Dominance Frontier** for each node in CFG.

Step 2: Regard alloca instructions as variables, its load instructions as uses, its store instructions as definitions.

Step 3: Insert **Phi-function** for each variable to its **Iterated Dominance Frontier(DF+)**.

Step 4: "Rename". **Replace uses** of load instruction. Remove alloca, load and store instructions.

- 搭建asm框架

1.8

```
int main()
{
    int a=1;
    int b=a+1;
    printlnInt(b);

    return 0;
}
```

1.16

- 今天完成printlnInt(1)
- 弄懂sp+2048的问题

其实这个问题可以之后再考虑

lui加载高20位， addi把低十二位放入 这个是将一个32位的立即数放入寄存器的方法

可以直接li

- li用法...?大体上li可能是一个非常强的指令 大胆用就可以
- 看函数调用的约定

1.17继续扩充指令 今天完成

我采用迭代器插入方法 调用内置库

现在状态很好欧里给

- 考虑function里面函数参数怎么处理
- 完成function 和callfuction部分 多余八个参数的两个地方都没有处理
- 有几个超过块大小的点修理一下
- 先完成最简单的那个函数 寄存器分配部分

关于iterator

我直接用一种方法写完regalloca

像这种代码是明显可以优化的吧

```
main:
    addi    sp, sp, -32
    sw     ra, 28(sp)
    sw     s0, 24(sp)
    addi    s0, sp, 32
    call    __cxx_global_var_init
    li     t0, 10000000
    sw     t0, -16(s0)
    lw     t0, -16(s0)
    mv     a0, t0
    call    printlnInt
    sw     zero, -12(s0)
    j     .BB0_1
```

1.18

- 函数递归
- 函数参数多余8
- 2048栈的大小
- 全局变量
- 其余指令 (优先 可以测试正确性) gep
- string&&gep
- global...?处理出错的点 全局变量的load store 不是很会处理qwqq 问一手

```
/*
Test Package: Codegen
Author: Admin
Time: 2020-02-03
Input:
==== input ====
==== end ====
Output:
==== output ====
==== end ====
ExitCode: 30
```

```
InstLimit: -1
Origin Package: Codegen Pretest-901
*/
int x;
int y;
int main() {
    x = 10;
    y = 20;
    printlnInt(x);
    int c = x+y;
    return c;
}
```

```
int main(){
    println("hello");
    return 0;
}
```

注意我现在的办法很暴力 就是add的时候如果一个是寄存器一个是数字 我也会把数字load出来的 而非使用addi

```
/*
Test Package: Codegen
Author: 14' Shichao xu
Time: 2020-01-25
Input:
==== input ====
==== end ====
Output:
==== output ====
1024
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-533
*/
int qpow(int a,int p,int mod) {
    int t = 1;
    int y = a;
    while(p>0){
        if((p&1) == 1)t=t*y % mod;
        y=y*y % mod;
        p=p / 2;
    }
    return t;
}
```

```
int main() {
```

```
    println(tostring(qpow(2,10,10000)));
    return 0;
}
```

这个点怎么debug

注意函数是有返回值的 我忘记写了okkk完成

计划晚上上oj

```
int kk(int a,int a1,int a2,int a3,int a4,int a5,int a6,int a7,int a8,int a9,int a10){

    return a+a1+a2+a3+a4+a5+a6+a7+a8+a9+a10;
}
int main(){
    printlnInt(kk(1,1,1,1,1,1,1,1,1,1));
    return 0;
}
```

gep可以改进 我直接array用一种写法 没有优化

bug

```
/*
Test Package: Codegen
Author: Haojun Mao
Time: 2020-01-25
Input:
==== input ====
1
2
3
4
==== end ====
Output:
==== output ====
4
1234
0000
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-532
*/
// input: 1 2 3 4
```

```
int[] a = new int[4];
int main()
{
    int[][] b = new int[4][];
    b[0] = a;
    printlnInt((b.size()));
}
```

```
// input: 1 2 3 4

int a=9;
int main()
{
    int[] b = new int[4];
    b[0] = a;
    printlnInt((b.size()));
}
```

todo

```
class mm {
int []q;
mm(){
q=new int[10];
q[1]=99;
}
;
int main(){
mm jjj=new mm;
printlnInt(jjj.q[1]);
}
```

gep class部分写错了 就是我们要在运行的过程中确定class的大小

```
何必折磨自己
class mm {
int q;
mm(){
q=9;
}
;
int main(){
mm jjj=new mm;
printlnInt(jjj.q);
}
```

```
/*
Test Package: Codegen
Author: 11' Tianxing Jin
Time: 2020-02-02
Input:
==== input ====
==== end ====
Output:
==== output ====
-66060719 -323398799 -743275679
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-570
```


六.optimize

图染色

```
#include<stdio.h>

int color[100];
bool ok(int k,int c[][100]) //判断顶点k的着色是否发生冲突
{
    int i,j;
    for(i=1;i<k;i++)
    {
        if(c[k][i]==1&&color[i]==color[k])
            return false;
    }
    return true;
}

void graphcolor(int n,int m,int c[][100])
{
    int i,k;
    for(i=1;i<=n;i++)
        color[i]=0;
    k=1;
    while(k>=1)
    {
        color[k]=color[k]+1;
```

```

while(color[k]<=m)
    if(ok(k,c)) break;
    else color[k]=color[k]+1; //搜索下一个颜色
if(color[k]<=m&&k==n)
{
    for(i=1;i<=n;i++)
        printf("%d ",color[i]);
    printf("\n");
}
else if(color[k]<=m&&k<n)
    k=k+1; //处理下一个顶点
else
{
    color[k]=0;
    k=k-1;//回溯
}
}
void main()
{
    int i,j,n,m;
    int c[100][100];//存储n个顶点的无向图的数组
    printf("输入顶点数n和着色数m:\n");
    scanf("%d %d",&n,&m);
    printf("输入无向图的邻接矩阵: \n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&c[i][j]);
    printf("着色所有可能的解:\n");
    graphcolor(n,m,c);
}

```

一般图染色问题的解法

0	0	1	0	1	0	0
0	0	0	0	0	1	0
1	0	0	0	1	0	0
0	0	0	0	0	0	1
1	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0

我把可以用物理寄存器的，放在一个栈里，然后再对栈里的用物理寄存器进行分配

没太懂qwq什么是可以用物理寄存器的



就是度少于寄存器数量的

instruction selector还有很大的优化空间

Background
From Liveness Analysis To Register Interference Graph (RIG)

Algorithm: Part I

- Compute live variables for each point:

```
graph TD; A["{a,c,f}"] --> B["{c,d,f}"]; B --> C["{c,e}"]; C --> D["{f := 2 * e}"]; D --> E["{c,f}"]; E --> F["{b}"]; F --> G["{b,c,f}"]; G --> H["{c,d,e,f}"]; H --> I["{b,c,e,f}"]; I --> J["{b}"];
```

Prof. Aiken CS 143 Lecture 16

Graph Coloring Example

- Consider the example RIG

Reference: Stanford CS143 Compilers.

从后往前进行liveness analysis

然后如果有生命周期的重合则进行连边

\生命周期也是以函数作为单位

- 关于调用者与被调用者
- caller callee寄存器怎么理解

Register	ABI Name	Saver	作用
x0	zero	—	硬编码恒为0
x1	ra	Caller	函数调用的返回地址
x2	sp	Callee	堆栈指针
x3	gp	—	全局指针
x4	tp	—	线程指针
x5–7	t0–2	Caller	临时寄存器/
x8	s0/fp	Callee	保存寄存器/帧指针
x9	s1	Callee	保存寄存器
x10–11	a0–1	Caller	函数参数/返回值
x12–17	a2–7	Caller	函数参数
x18–27	s2–11	Callee	保存寄存器
x28–31	t3–6	Caller	临时寄存器

就是caller-saved callee-saved的意思

日志

1.28 学习和写liveanalysis

结点是一条指令

一个结点的def是该结点定值的变量的集合 类似可以定义一个结点的Use

在一条边上是活跃的定义：存在一条从这条边通向该变量一个Use的有向路径

1.29

因此需要注意的是，在分配寄存器时，能优先分配 caller 寄存器的话应该优先分配 caller 寄存器，否则在遇到生命周期跨过 call 的寄存器时会可能导致 callee 寄存器不够用而存放在栈中的情况。

活性分析以块作为单位 由于每一个块都没有流的变化 所以可以这样子操作

1.30

todo 用stack先测试一下liveanalysis

precolored是只有物理寄存器

intialed是虚拟寄存器

`retainAll(Collection c)`

两个集合求交集，只保留交集数据

集合取交集的接口

2.1

当

```
for (RISCVBasicBlock tmpBlock = curFunc.EntranceBlock;
    tmpBlock != null; tmpBlock = tmpBlock.nextBlock) {
    for (tmpInst = tmpBlock.HeadInst;
        tmpInst != null; tmpInst = tmpInst.nextInst) {
        if (tmpInst instanceof RISCVmvInst) {
            if (((RISCVmvInst) tmpInst).rd.color == ((RISCVmvInst) tmpInst).rs1.color) {
                tmpInst.removeInst(tmpBlock);
            }
        }
    }
}
```

这种情况需要处理

2.2

一个是函数刚刚进来的时候mv指令过多 怎么去除...?

- ```
/*
Test Package: Codegen
Author: Pikachu
Time: 2020-02-03
Input:
==== input ====
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 0 0 0
0 0 0 0 0
==== end ====
Output:
==== output ====
1
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codeforces 263A #48540584
*/
int n;
int r;
int c;
int i;
int j;
int abs(int c){
 if(c>0) return c;
 return -c;
}

int main()
{
 // int n,r,c,i,j;
 for (i=0;i<5;i++)
 for (j=0;j<5;j++)
 {
```

```

 n = getInt();
 //scanf("%d", &n);
 if (n==1)
 {r=i;c=j;}
 printInt(abs(2-r) + abs(2-c));
 // printf("%d",abs(2-r)+abs(2-c));
 return 0;
}

```

找到的唯一还能debug的数据点

```

/*
Test Package: codegen
Author: Pikachu
Time: 2020-02-03
Input:
==== input ====
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 0 0 0
0 0 0 0 0
==== end ====
Output:
==== output ====
1
==== end ====
ExitCode: 0
InstLimit: -1
Origin Package: Codeforces 263A #48540584
*/
int n;
int r;
int c;
int i;
int j;
int abs(int c){
 printlnInt(c);
 return c;
}

int main()
{
 r=2;
 c=1;
 printlnInt(abs(2-r) + abs(2-c));
 return 0;
}

```

简化后的数据点

可以用单步测试指令来观察

麻了 我超本质上就是去掉那堆乱七八糟的mv操作

e6的神秘bug

Interpretation finished in 505 ms

⇒ Test Finished

→ Judger: Judging

Start Time: 2022.02.02 00:24:15

=Output: Passed=

=Exit code: Failed=

Expected: [0]

Actual: [8]

exit code failed 神秘 他的输出是对的

2.2

保命数据点

- /\*  
Test Package: Codegen  
Author: Pikachu  
Time: 2020-02-03  
Input:  
==== input ====  
0 0 0 0 0  
0 0 0 0 0  
0 1 0 0 0  
0 0 0 0 0  
0 0 0 0 0  
==== end ====  
Output:  
==== output ====  
1  
==== end ====  
ExitCode: 0  
InstLimit: -1  
Origin Package: Codeforces 263A #48540584  
\*/  
int n;  
int r;  
int c;  
int i;  
int j;  
int abs(int c){  
 if(c>0) return c;  
 return -c;  
}

```

int main()
{
 r=2;c=1;
 printInt(abs(2-r) + abs(2-c));
 // printf("%d",abs(2-r)+abs(2-c));
 return 0;
}

```

直觉告诉我出错的地方在零号寄存器

2.3

```

int gcd(int x, int y, int j0 , int j1 , int j2 , int j3 , int j4 , int j5 , int
j6 , int j7 , int j8 , int j9 , int j10 , int j11) {
 return (x+y+j0 +j1 +j2 +j3 +j4 +j5 +j6 +j7 +j8 +j9 +j10 +j11);
}
int main() {
 printlnInt((gcd(1,1, 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1));
 return 0;
}

```

造出的一个错误数据 借着这个对拍debug

```

int gcd(int x, int y, int j0 , int j1 , int j2 , int j3 , int j4 , int j5 , int
j6 , int j7 , int j8 , int j9 , int j10 , int j11 ,int j12,int j13,int j14,int
j15,int j16,int j17) {
 return (x+y+j0 +j1 +j2 +j3 +j4 +j5 +j6 +j7 +j8 +j9 +j10 +j11
+j12+j13+j14+j15+j16+j17);
}
int main() {
 printlnInt((gcd(1,1, 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1,1,1,1,1,1
)));
 return 0;
}

```

新的错误数据点

```

int gcd(int x, int y, int j0 , int j1 , int j2 , int j3 , int j4 , int j5 , int
j6 , int j7 , int j8 , int j9 , int j10 , int j11 , int j12 , int j13 , int j14 ,
int j15 , int j16 , int j17 , int j18 , int j19 , int j20 , int j21 , int j22 ,
int j23 , int j24 , int j25 , int j26 , int j27 , int j28 , int j29) {
 return y;
}
int main() {
 printlnInt((gcd(10,1, 0 , 2 , 4 , 6 , 8 , 10 , 12 , 14 , 16 , 18 , 20 , 22 ,
24 , 26 , 28 , 30 , 32 , 34 , 36 , 38 , 40 , 42 , 44 , 46 , 48 , 50 , 52 , 54 ,
56 , 58)));
 return 0;
}

```

为什么color会分配成zero ?

解决

新的数据点

这个数据点还有价值 问题就是这个循环的次数很多要进行很多次rewrite

```
int gcd(int x, int y, int j0 , int j1 , int j2 , int j3 , int j4 , int j5 , int j6 , int j7 , int j8 , int j9 , int j10 , int j11 , int j12 , int j13 , int j14 , int j15 , int j16 , int j17 , int j18 , int j19 , int j20 , int j21 , int j22 , int j23 , int j24 , int j25) {
 return (j0 +j1 +j2 +j3 +j4 +j5 +j6 +j7 +j8 +j9 +j10 +j11 +j12 +j13 +j14 +j15 +j16 +j17 +j18 +j19 +j20 +j21 +j22 +j23 +j24 +j25) ;
}
int main() {
 println(toString(gcd(10,1, 0 , 2 , 4 , 6 , 8 , 10 , 12 , 14 , 16 , 18 , 20 , 22 , 24 , 26 , 28 , 30 , 32 , 34 , 36 , 38 , 40 , 42 , 44 , 46 , 48 , 50)));
 return 0;
}
```

| Phase      | Judge Result                        | Rate    |
|------------|-------------------------------------|---------|
| 0-Build    | <div style="width: 100%;">1</div>   | 100.0%  |
| 1-Semantic | <div style="width: 100%;">190</div> | 100.00% |
| 2-Codegen  | <div style="width: 100%;">85</div>  | 100.00% |

2.4

一个是spillselect选择问题

```
=====
final namej5_para
1 ****
vr name: virtual_register_spill_use
2.0
28|
vr name: j13_para
4.0
29
vr name: j3_para
4.0
31
vr name: j22_para
4.0
29
vr name: j7_para
4.0
29
vr name: j17_para
4.0
29
vr name: j20_para
4.0
29
```

下面这个可以先放一下 最后没弄出来也没事的

一个是正确性问题 问郑老板

一个是 继续优化问题 xch

2.4

performance

|     |            |                  |                     |       |             |                     |
|-----|------------|------------------|---------------------|-------|-------------|---------------------|
| 283 | 3-Optimize | hidden-1.mx      | <span>accept</span> | 1.39s | 3594549618  | 2022.02.03 18:34:59 |
| 284 | 3-Optimize | hidden-2.mx      | <span>fail</span>   | 1.53s | 39229255    | 2022.02.03 18:35:02 |
| 285 | 3-Optimize | hidden-3.mx      | <span>accept</span> | 1.24s | 16768976417 | 2022.02.03 18:35:15 |
| 286 | 3-Optimize | hidden-4.mx      | <span>accept</span> | 1.08s | 1348580410  | 2022.02.03 18:35:18 |
| 287 | 3-Optimize | hidden-5.mx      | <span>accept</span> | 1.12s | 782715825   | 2022.02.03 18:35:20 |
| 288 | 3-Optimize | hidden-6.mx      | <span>fail</span>   | 2.93s | 21474800000 | 2022.02.03 18:35:36 |
| 289 | 3-Optimize | hidden-7.mx      | <span>accept</span> | 1.07s | 365167295   | 2022.02.03 18:35:38 |
| 290 | 3-Optimize | hidden-8.mx      | <span>accept</span> | 0.72s | 467608612   | 2022.02.03 18:35:41 |
| 291 | 3-Optimize | hidden-9.mx      | <span>accept</span> | 1.83s | 11063272969 | 2022.02.03 18:35:50 |
| 292 | 3-Optimize | hidden-10.mx     | <span>fail</span>   | 1.53s | 43214600622 | 2022.02.03 18:36:21 |
| 293 | 3-Optimize | hidden-11.mx     | <span>accept</span> | 9.26s | 4010762546  | 2022.02.03 18:36:42 |
| 294 | 3-Optimize | hidden-12-big.mx | <span>accept</span> | 1.98s | 5612601239  | 2022.02.03 18:36:47 |

2.5

myperformance today

|     |            |                  |        |       |             |                     |
|-----|------------|------------------|--------|-------|-------------|---------------------|
| 283 | 3-Optimize | hidden-1.mx      | accept | 1.37s | 3594549618  | 2022.02.05 15:56:00 |
| 284 | 3-Optimize | hidden-2.mx      | fail   | 2.10s | 39229255    | 2022.02.05 15:56:03 |
| 285 | 3-Optimize | hidden-3.mx      | accept | 1.38s | 16768976417 | 2022.02.05 15:56:18 |
| 286 | 3-Optimize | hidden-4.mx      | accept | 1.07s | 1348580410  | 2022.02.05 15:56:21 |
| 287 | 3-Optimize | hidden-5.mx      | accept | 1.57s | 782715825   | 2022.02.05 15:56:24 |
| 288 | 3-Optimize | hidden-6.mx      | fail   | 2.78s | 21474800000 | 2022.02.05 15:56:38 |
| 289 | 3-Optimize | hidden-7.mx      | accept | 1.27s | 365167295   | 2022.02.05 15:56:41 |
| 290 | 3-Optimize | hidden-8.mx      | accept | 0.77s | 467608612   | 2022.02.05 15:56:44 |
| 291 | 3-Optimize | hidden-9.mx      | accept | 1.82s | 11063272969 | 2022.02.05 15:56:52 |
| 292 | 3-Optimize | hidden-10.mx     | fail   | 1.57s | 43214600622 | 2022.02.05 15:57:22 |
| 293 | 3-Optimize | hidden-11.mx     | accept | 8.97s | 4010762546  | 2022.02.05 15:57:44 |
| 294 | 3-Optimize | hidden-12-big.mx | accept | 2.48s | 5612601239  | 2022.02.05 15:57:50 |

a wrong point

```
/*
Test Package: Optim
Author: zhuolin Yang, 15
Input:
==== input ====
3

==== end ====
Output:
==== output ====
2
4
846

==== end ====
ExitCode: 0
InstLimit: -1
*/
int Mod;
int[][] p;
int[] res;
int[][] ksm;
int[] prime;
int tot;
int[] v;
int[][] q;
int[][][] g;
int[][][] sum;
int[] m;
int[] b;
int[][] Comb;

int C;
int M;
```

```

int N;
int[] fn;
int[] fc;
int[][] fm;

void init() {
 tot = 0;
 C = 0;
 M = 0;
 N = 0;
 int i;
 int j;
 int k;
 g = new int[6][][][];
 for (i = 0; i < 6; i++) {
 g[i] = new int[100001][];
 for (j = 0; j < 100001; j++) {
 g[i][j] = new int[15];
 for (k = 0; k < 15; k++)
 g[i][j][k] = 0;
 }
 }
 Sum = new int[6][][][];
 for (i = 0; i < 6; i++) {
 Sum[i] = new int[100001][];
 for (j = 0; j < 100001; j++) {
 Sum[i][j] = new int[15];
 for (k = 0; k < 15; k++)
 Sum[i][j][k] = 0;
 }
 }
 fm = new int[1001][];
 for (i = 0; i < 1001; i++) {
 fm[i] = new int[13];
 for (j = 0; j < 13; j++)
 fm[i][j] = 0;
 }
 ksm = new int[100001][];
 for (i = 0; i < 100001; i++) {
 ksm[i] = new int[21];
 for (j = 0; j < 21; j++)
 ksm[i][j] = 0;
 }
 p = new int[21][];
 for (i = 0; i < 21; i++) {
 p[i] = new int[21];
 for (j = 0; j < 21; j++)
 p[i][j] = 0;
 }
 q = new int[21][];
 for (i = 0; i < 21; i++) {
 q[i] = new int[100001];
 for (j = 0; j < 100001; j++)
 q[i][j] = 0;
 }
 Comb = new int[100001][];
 for (i = 0; i < 100001; i++) {
 Comb[i] = new int[21];
 }
}

```

```

 for (j = 0; j < 21; j++)
 Comb[i][j] = 0;
 }
 fn = new int[1001];
 for (i = 0; i < 1001; i++) fn[i] = 0;
 fc = new int[1001];
 for (i = 0; i < 1001; i++) fc[i] = 0;
 m = new int[1001];
 for (i = 0; i < 1001; i++) m[i] = 0;
 res = new int[1001];
 for (i = 0; i < 1001; i++) res[i] = 0;
 b = new int[1001];
 for (i = 0; i < 1001; i++) b[i] = 0;
 v = new int[100001];
 for (i = 0; i < 100001; i++) v[i] = 0;
 prime = new int[100001];
 for (i = 0; i < 100001; i++) prime[i] = 0;
}
int Ksm(int P, int x)
{
 if (x == 0) return 1;
 if (x == 1) return P % Mod;
 int tmp;
 tmp = Ksm(P, x >> 1);
 if (x % 2 == 1) return tmp * tmp % Mod * P % Mod;
 else return tmp * tmp % Mod;
}

void Calculate_p()
{
 int i;
 int j;
 p[0][0] = 1;
 p[1][1] = 1;
 p[1][0] = Mod - 1;
 for (i = 2; i <= C - 2; i++)
 {
 int tmp;
 tmp = Ksm(i, Mod - 2);
 for (j = 0; j < i; j++)
 p[i][j + 1] = p[i - 1][j];
 for (j = 0; j <= i; j++)
 p[i][j] = (p[i][j] - p[i - 1][j] * i % Mod + Mod) * tmp % Mod;
 }
}
void Euler(int x)
{
 tot = 0;
 q[x][1] = 1;
 int i;
 int j;
 for (i = 0; i < 100001; i++) v[i] = 0;
 for (i = 2; i <= M; i++)
 {
 if (v[i] == 0){
 prime[++ tot] = i;
 q[x][i] = (ksm[i][x] + Mod - 1) % Mod;
 }
 }
}

```

```

 }
 for (j = 1; j <= tot && prime[j] * i <= M; j++)
 {
 v[prime[j] * i] = 1;
 if (i % prime[j] == 0)
 {
 q[x][i * prime[j]] = q[x][i] * ksm[prime[j]][x] % Mod;
 break;
 }
 else q[x][i * prime[j]] = q[x][i] * q[x][prime[j]] % Mod;
 }
}
void calculate_q()
{
 int i;
 for (i = 0; i <= c - 2; i++) Euler(i);
}
void calculate_ksm()
{
 int i;
 int j;
 for (i = 1; i <= M; i++)
 {
 ksm[i][0] = 1;
 for (j = 1; j <= c - 2; j++)
 ksm[i][j] = ksm[i][j - 1] * i % Mod;
 }
}
void calculate_G()
{
 calculate_ksm(); calculate_p(); calculate_q();
 int i;
 int j;
 int k;
 for (i = 1; i <= M; i++)
 for (j = 2; j <= c; j++)
 {
 for (k = 0; k <= j - 2; k++)
 g[0][i][j] = (g[0][i][j] + p[j - 2][k] * q[k][i]) % Mod;
 for (k = 1; k <= N; k++)
 g[k][i][j] = g[k - 1][i][j] * i % Mod;
 }
 for (k = 0; k <= N; k++)
 {
 for (i = 2; i <= c; i++)
 for (j = 1; j <= M; j++)
 {
 sum[k][j][i] = sum[k][j - 1][i] + g[k][j][i];
 if (sum[k][j][i] >= Mod) sum[k][j][i] = sum[k][j][i] - Mod;
 }
 }
}
void calculate_comb()
{
 int i;
 int j;
 for (i = 0; i <= M; i++)

```

```

Comb[i][0] = 1;
for (i = 1; i <= M; i++)
 for (j = 1; j <= C; j++)
 {
 Comb[i][j] = Comb[i - 1][j] + Comb[i - 1][j - 1];
 if (Comb[i][j] >= Mod) Comb[i][j] = Comb[i][j] - Mod;
 }
}
int main()
{
 int T;
 int n;
 int c;

 T = getInt();
 init();
 fn[1] = 2; fc[1] = 3;
 fm[1][1] = 3; fm[1][2] = 4;
 fn[2] = 3; fc[2] = 3;
 fm[2][1] = 3; fm[2][2] = 4; fm[2][3] = 4;
 fn[3] = 4; fc[3] = 4;
 fm[3][1] = 5; fm[3][2] = 7; fm[3][3] = 8; fm[3][4] = 9;
 C = 4; M = 9;
 N = 4;
 Mod = 10007;
 calculate_G(); calculate_comb();
 int rp;
 for (rp = 1; rp <= T; rp++)
 {
 n = fn[rp];
 c = fc[rp];
 int ii;
 for (ii = 1; ii <= n; ii++)
 m[ii] = fm[rp][ii];
 if (n == 1) println(toString(Comb[m[1]][c]));
 else
 {
 int ans;
 int l;
 int r;
 int temp;
 ans = 0;
 temp = 0;
 l = 0;
 r = 1;

 for (; r <= m[1]; r++)
 {
 int tmp;
 tmp = (m[1] / (m[1] / r));
 int i;
 for (i = 2; i <= n; i++) {
 int get;
 get = m[i] / (m[i] / r);
 if (get < tmp) tmp = get;
 }
 if (m[1] < tmp) tmp = m[1];
 r = tmp;
 }
 if (ans <= 0) ans = tmp;
 else if (ans > tmp) ans = tmp;
 else if (ans < tmp) ans = ans * r + tmp;
 else if (ans > tmp) ans = ans * r - tmp;
 else if (ans == tmp) ans = ans * r;
 }
 }
}

```

```

 b[0] = 1;
 for (i = 1; i <= n; i++)
 {
 int tmp1;
 int tmp2;
 int tmp3;
 tmp1 = (m[i] / r) % Mod;
 tmp2 = tmp1 * (tmp1 + 1) / 2 % Mod;
 tmp3 = m[i] * tmp1 % Mod;
 int j;
 for (j = 0; j < i; j++)
 res[j + 1] = b[j] * (Mod - tmp2) % Mod;
 res[0] = 0;
 for (j = 0; j < i; j++)
 res[j] = (res[j] + tmp3 * b[j]) % Mod;
 for (j = 0; j <= i; j++) b[j] = res[j];
 }
 for (i = 0; i <= n; i++)
 ans = (ans + b[i] * (sum[i][r][c] - sum[i][l][c])) % Mod;
 l = r;
 }
 if (ans < 0) ans = ans + Mod;
 println(tostring(ans));
}
return 0;
}

```

**=Report=**

```

exit code: 0
memory leak: 1
instructions count: 2472 line(s)
global memory: 9920 Byte(s)
time: 9268492806
instructions:
simple = 400313203 (including unconditional jump)
mul = 71138576
cache = 0
br = 27621804
div = 284
mem = 127784450 (a.k.a cache miss)
libcIO = 4
libcMem = 1440965

```

Start time: 2022-02-17 10:44:00

-----  
=Report=

```
exit code: 0
memory leak: 1
instructions count: 1496 line(s)
global memory: 6048 Byte(s)
time: 22566377
instructions:
simple = 834002 (including unconditional jump)
mul = 55395
cache = 0
br = 39368
div = 0
mem = 299953 (a.k.a cache miss)
libcIO = 26000
libcMem = 2616
```

=Simulator Stderr=

hyj 6

-----  
=Report=

```
exit code: 0
memory leak: 1
instructions count: 2891 line(s)
global memory: 11600 Byte(s)
time: 7677037514
instructions:
simple = 482586784 (including unconditional jump)
mul = 72541091
cache = 0
br = 27622325
div = 700
mem = 101566537 (a.k.a cache miss)
libcIO = 7
libcMem = 1430026
```

xt 2

--report--

```
exit code: 0
memory leak: 1
instructions count: 1511 line(s)
global memory: 6112 Byte(s)
time: 957299103
instructions:
simple = 41156124 (including unconditional jump)
mul = 0
cache = 0
br = 1404839
div = 0
mem = 14025887 (a.k.a cache miss)
libcIO = 36000
libcMem = 38621
```

xt 6

```
exit code: 0
memory leak: 1
instructions count: 2478 line(s)
global memory: 9952 Byte(s)
time: 7485013705
instructions:
simple = 308011230 (including unconditional jump)
mul = 538
cache = 0
br = 27622325
div = 665
mem = 105769807 (a.k.a cache miss)
libcIO = 7
libcMem = 1458971
```

lt 2

```
exit code: 0
memory leak: 1
instructions count: 1731 line(s)
global memory: 6992 Byte(s)
time: 1559130277
instructions:
simple = 45259392 (including unconditional jump)
mul = 2956948
cache = 0
br = 1444385
div = 55480
mem = 23062684 (a.k.a cache miss)
libcIO = 26000
libcMem = 96628
```

lht 6

-----

=Report=

```
exit code: 0
memory leak: 1
instructions count: 2787 line(s)
global memory: 11184 Byte(s)
time: 7355495311
instructions:
simple = 441253278 (including unconditional jump)
mul = 69738687
cache = 0
br = 27622336
div = 700
mem = 97363398 (a.k.a cache miss)
libcIO = 4
libcMem = 1430041
```

2.15

```
/*
Test Package: Optim
Author: Zhuolin Yang, 15
Input:
==== input ====
3

==== end ====
Output:
==== output ====
2
4
846

==== end ====
ExitCode: 0
InstLimit: -1
*/
```

```
int Mod;
int[][] p;
int[] res;
int[][] ksm;
int[] prime;
int tot;
int[] v;
int[][] q;
int[][][] g;
int[][][] sum;
int[] m;
int[] b;
int[][] Comb;
```

```

int C;
int M;
int N;
int[] fn;
int[] fc;
int[][] fm;

void init() {
 tot = 0;
 C = 0;
 M = 0;
 N = 0;
 int i;
 int j;
 int k;
 g = new int[6][][];
 for (i = 0; i < 6; i++) {
 g[i] = new int[100001]++;
 for (j = 0; j < 100001; j++) {
 g[i][j] = new int[15];
 for (k = 0; k < 15; k++)
 g[i][j][k] = 0;
 }
 }
 Sum = new int[6][][];
 for (i = 0; i < 6; i++) {
 Sum[i] = new int[100001]++;
 for (j = 0; j < 100001; j++) {
 Sum[i][j] = new int[15];
 for (k = 0; k < 15; k++)
 Sum[i][j][k] = 0;
 }
 }
 fm = new int[1001]++;
 for (i = 0; i < 1001; i++) {
 fm[i] = new int[13];
 for (j = 0; j < 13; j++)
 fm[i][j] = 0;
 }
 ksm = new int[100001]++;
 for (i = 0; i < 100001; i++) {
 ksm[i] = new int[21];
 for (j = 0; j < 21; j++)
 ksm[i][j] = 0;
 }
 p = new int[21]++;
 for (i = 0; i < 21; i++) {
 p[i] = new int[21];
 for (j = 0; j < 21; j++)
 p[i][j] = 0;
 }
 q = new int[21]++;
 for (i = 0; i < 21; i++) {
 q[i] = new int[100001];
 for (j = 0; j < 100001; j++)
 q[i][j] = 0;
 }
 Comb = new int[100001]++;
}

```

```

for (i = 0; i < 100001; i++) {
 Comb[i] = new int[21];
 for (j = 0; j < 21; j++)
 Comb[i][j] = 0;
}
fn = new int[1001];
for (i = 0; i < 1001; i++) fn[i] = 0;
fc = new int[1001];
for (i = 0; i < 1001; i++) fc[i] = 0;
m = new int[1001];
for (i = 0; i < 1001; i++) m[i] = 0;
res = new int[1001];
for (i = 0; i < 1001; i++) res[i] = 0;
b = new int[1001];
for (i = 0; i < 1001; i++) b[i] = 0;
v = new int[100001];
for (i = 0; i < 100001; i++) v[i] = 0;
prime = new int[100001];
for (i = 0; i < 100001; i++) prime[i] = 0;
}
int Ksm(int P, int x)
{
 if (x == 0) return 1;
 if (x == 1) return P % Mod;
 int tmp;
 tmp = Ksm(P, x >> 1);
 if (x % 2 == 1) return tmp * tmp % Mod * P % Mod;
 else return tmp * tmp % Mod;
}

void calculate_p()
{
 int i;
 int j;
 p[0][0] = 1;
 p[1][1] = 1;
 p[1][0] = Mod - 1;
 for (i = 2; i <= c - 2; i++)
 {
 int tmp;
 tmp = Ksm(i, Mod - 2);
 for (j = 0; j < i; j++)
 p[i][j + 1] = p[i - 1][j];
 for (j = 0; j <= i; j++)
 p[i][j] = (p[i][j] - p[i - 1][j] * i % Mod + Mod) * tmp % Mod;
 }
}
void Euler(int x)
{
 tot = 0;
 q[x][1] = 1;
 int i;
 int j;
 for (i = 0; i < 100001; i++) v[i] = 0;
 for (i = 2; i <= M; i++)
 {
 if (v[i] == 0){

```

```

 prime[++ tot] = i;
 q[x][i] = (ksm[i][x] + Mod - 1) % Mod;
 }
 for (j = 1; j <= tot && prime[j] * i <= M; j++)
 {
 v[prime[j] * i] = 1;
 if (i % prime[j] == 0)
 {
 q[x][i * prime[j]] = q[x][i] * ksm[prime[j]][x] % Mod;
 break;
 }
 else q[x][i * prime[j]] = q[x][i] * q[x][prime[j]] % Mod;
 }
}
void calculate_q()
{
 int i;
 for (i = 0; i <= c - 2; i++) Euler(i);
}
void calculate_Ksm()
{
 int i;
 int j;
 for (i = 1; i <= M; i++)
 {
 ksm[i][0] = 1;
 for (j = 1; j <= c - 2; j++)
 ksm[i][j] = ksm[i][j - 1] * i % Mod;
 }
}
void calculate_G()
{
 calculate_Ksm(); calculate_p(); calculate_q();
 int i;
 int j;
 int k;
 for (i = 1; i <= M; i++)
 for (j = 2; j <= c; j++)
 {
 for (k = 0; k <= j - 2; k++)
 g[0][i][j] = (g[0][i][j] + p[j - 2][k] * q[k][i]) % Mod;
 for (k = 1; k <= N; k++)
 g[k][i][j] = g[k - 1][i][j] * i % Mod;
 }
 for (k = 0; k <= N; k++)
 {
 for (i = 2; i <= c; i++)
 for (j = 1; j <= M; j++)
 {
 sum[k][j][i] = sum[k][j - 1][i] + g[k][j][i];
 if (sum[k][j][i] >= Mod) sum[k][j][i] = sum[k][j][i] - Mod;
 }
 }
}
void calculate_Comb()
{
 int i;

```

```

int j;
for (i = 0; i <= M; i++)
 Comb[i][0] = 1;
for (i = 1; i <= M; i++)
 for (j = 1; j <= C; j++)
 {
 Comb[i][j] = Comb[i - 1][j] + Comb[i - 1][j - 1];
 if (Comb[i][j] >= Mod) Comb[i][j] = Comb[i][j] - Mod;
 }
}
int main()
{
 int T;
 int n;
 int c;

 T = getInt();
 init();
 fn[1] = 2; fc[1] = 3;
 fm[1][1] = 3; fm[1][2] = 4;
 fn[2] = 3; fc[2] = 3;
 fm[2][1] = 3; fm[2][2] = 4; fm[2][3] = 4;
 fn[3] = 4; fc[3] = 4;
 fm[3][1] = 5; fm[3][2] = 7; fm[3][3] = 8; fm[3][4] = 9;
 C = 4; M = 9;
 N = 4;
 Mod = 10007;
 Calculate_G(); calculate_Comb();
 int rp;
 for (rp = 1; rp <= T; rp++)
 {
 n = fn[rp];
 c = fc[rp];
 int ii;
 for (ii = 1; ii <= n; ii++)
 m[ii] = fm[rp][ii];
 if (n == 1) println(toString(Comb[m[1]][c]));
 else
 {
 int ans;
 int l;
 int r;
 int temp;
 ans = 0;
 temp = 0;
 l = 0;
 r = 1;

 for (; r <= m[1]; r++)
 {
 int tmp;
 tmp = (m[1] / (m[1] / r));
 int i;
 for (i = 2; i <= n; i++) {
 int get;
 get = m[i] / (m[i] / r);
 if (get < tmp) tmp = get;
 }
 }
 }
 }
}

```

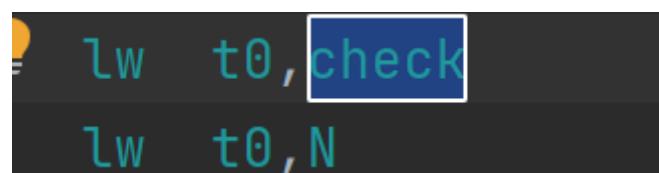
```

 if (m[1] < tmp) tmp = m[1];
 r = tmp;
 b[0] = 1;
 for (i = 1; i <= n; i++)
 {
 int tmp1;
 int tmp2;
 int tmp3;
 tmp1 = (m[i] / r) % Mod;
 tmp2 = tmp1 * (tmp1 + 1) / 2 % Mod;
 tmp3 = m[i] * tmp1 % Mod;
 int j;
 for (j = 0; j < i; j++)
 res[j + 1] = b[j] * (Mod - tmp2) % Mod;
 res[0] = 0;
 for (j = 0; j < i; j++)
 res[j] = (res[j] + tmp3 * b[j]) % Mod;
 for (j = 0; j <= i; j++) b[j] = res[j];
 }
 for (i = 0; i <= n; i++)
 ans = (ans + b[i] * (sum[i][r][c] - sum[i][l][c])) % Mod;
 l = r;
 }
 if (ans < 0) ans = ans + Mod;
 println(toString(ans));
}
return 0;
}

```

## peephole

A classic example pattern is a store followed by a load from the same location. The load can be replaced by a copy.

$$\begin{array}{ccc} \text{storeAI } r_1 & \Rightarrow & r_{\text{arp}}, 8 \\ \text{loadAI } r_{\text{arp}}, 8 & \Rightarrow & r_{15} \end{array} \Rightarrow \begin{array}{ccc} \text{storeAI } r_1 & \Rightarrow & r_{\text{arp}}, 8 \\ \text{i2i} & & r_1 \Rightarrow r_{15} \end{array}$$


这个是可以去掉的

## 2.8 突破 造出错误的数据点 debugiing

这个是可以优化掉的

```
li t1,1
and t1,t2,t1
li a1,1
xor t1,t1,a1
```

```
/*
Test Package: Codegen
Author: Yunwei Ren
Input:
==== input ====
100 1966
0 11 20
0 12 42
0 14 89
0 17 11
0 18 36
0 22 49
0 32 90
0 42 59
0 43 69
0 71 29
0 86 18
0 91 73
0 92 52
0 97 43
0 99 2
1 3 20
1 11 10
1 13 16
1 15 6
1 16 79
1 20 95
1 22 4
1 25 14
1 26 77
1 32 69
1 42 94
1 44 48
1 46 13
1 50 71
1 53 32
1 56 95
1 63 27
1 66 92
1 72 56
1 73 41
1 75 1
1 87 51
2 4 56
2 22 12
2 34 85
2 49 78
2 61 44
2 68 70
```

2 73 86  
2 78 72  
2 83 92  
2 85 50  
2 86 24  
2 90 62  
2 94 97  
3 6 4  
3 11 57  
3 24 35  
3 26 50  
3 30 99  
3 34 55  
3 43 97  
3 47 92  
3 52 39  
3 53 34  
3 58 95  
3 59 62  
3 65 58  
3 66 95  
3 75 76  
3 78 49  
3 84 33  
3 85 45  
3 92 88  
4 13 63  
4 17 3  
4 23 33  
4 24 31  
4 28 13  
4 32 73  
4 39 82  
4 44 67  
4 55 21  
4 64 90  
4 68 90  
4 76 99  
4 80 34  
5 7 5  
5 15 49  
5 19 43  
5 21 59  
5 22 46  
5 24 9  
5 29 54  
5 34 39  
5 35 28  
5 39 3  
5 48 55  
5 51 14  
5 55 50  
5 63 60  
5 68 56  
5 81 73  
5 82 71  
5 88 57  
6 13 7

6 14 51  
6 15 29  
6 23 82  
6 29 71  
6 32 97  
6 36 46  
6 40 76  
6 42 99  
6 43 84  
6 52 53  
6 57 67  
6 61 33  
6 76 26  
6 84 88  
7 9 74  
7 18 37  
7 29 27  
7 34 12  
7 43 92  
7 49 63  
7 62 30  
7 71 74  
7 76 62  
7 77 14  
7 78 88  
7 92 64  
7 95 11  
7 96 71  
7 99 72  
8 11 87  
8 12 4  
8 28 1  
8 35 36  
8 45 72  
8 46 93  
8 47 98  
8 48 43  
8 58 83  
8 60 75  
8 63 32  
8 66 30  
8 69 53  
8 70 64  
8 76 71  
8 77 18  
8 78 90  
8 86 82  
8 95 18  
8 96 86  
8 99 68  
9 2 76  
9 13 33  
9 15 68  
9 22 64  
9 36 21  
9 46 93  
9 47 68  
9 49 39

9 52 93  
9 69 43  
9 77 9  
9 80 22  
9 85 7  
9 90 23  
9 95 44  
9 98 11  
10 4 50  
10 6 68  
10 13 89  
10 16 16  
10 19 51  
10 23 12  
10 24 20  
10 25 67  
10 30 65  
10 32 85  
10 46 64  
10 49 43  
10 52 63  
10 62 16  
10 64 91  
10 66 78  
10 78 80  
10 83 46  
10 84 84  
10 85 77  
10 88 32  
10 99 87  
11 2 17  
11 3 53  
11 4 18  
11 6 59  
11 13 95  
11 23 97  
11 25 53  
11 27 59  
11 29 31  
11 30 52  
11 31 6  
11 38 87  
11 39 29  
11 40 10  
11 61 18  
11 66 91  
11 75 5  
11 76 37  
11 78 38  
11 87 68  
11 88 29  
11 97 41  
11 99 12  
12 5 32  
12 14 1  
12 15 7  
12 22 10  
12 43 23

12 44 50  
12 51 62  
12 55 70  
12 56 32  
12 59 98  
12 62 37  
12 63 66  
12 65 67  
12 70 19  
12 71 21  
12 74 95  
12 82 74  
12 84 8  
12 89 93  
13 0 44  
13 20 74  
13 22 27  
13 24 68  
13 37 68  
13 38 27  
13 44 27  
13 47 32  
13 52 63  
13 56 1  
13 64 95  
13 65 1  
13 74 43  
13 75 1  
13 76 40  
13 80 51  
13 82 71  
13 83 58  
13 84 74  
13 91 63  
13 94 43  
14 7 30  
14 11 69  
14 33 25  
14 38 40  
14 40 94  
14 43 16  
14 48 52  
14 59 59  
14 64 42  
14 66 43  
14 77 83  
14 78 21  
14 79 70  
14 89 13  
14 91 38  
15 14 43  
15 21 66  
15 23 67  
15 25 89  
15 28 88  
15 29 32  
15 32 26  
15 33 77

15 37 78  
15 39 37  
15 51 70  
15 53 57  
15 57 63  
15 62 84  
15 64 21  
15 70 39  
15 74 25  
15 75 58  
15 81 88  
15 87 9  
15 93 11  
15 94 74  
15 99 8  
16 2 99  
16 4 24  
16 5 44  
16 10 67  
16 17 46  
16 18 83  
16 28 6  
16 35 38  
16 36 76  
16 44 41  
16 45 60  
16 49 42  
16 55 87  
16 61 85  
16 63 34  
16 65 88  
16 71 25  
16 84 10  
16 90 24  
16 96 15  
16 98 87  
17 3 94  
17 9 30  
17 21 44  
17 30 29  
17 32 18  
17 34 20  
17 35 14  
17 44 62  
17 48 84  
17 57 74  
17 60 95  
17 66 76  
17 68 84  
17 71 34  
17 81 50  
17 84 13  
17 89 18  
17 91 27  
17 94 1  
18 0 13  
18 2 85  
18 7 47

18 10 78  
18 16 22  
18 21 51  
18 23 44  
18 27 9  
18 30 42  
18 33 49  
18 44 54  
18 55 94  
18 59 52  
18 60 74  
18 63 30  
18 68 91  
18 74 71  
18 75 47  
18 76 52  
18 80 74  
18 81 11  
18 82 96  
18 90 97  
18 94 18  
18 96 93  
18 98 62  
19 2 51  
19 10 67  
19 14 80  
19 23 59  
19 30 80  
19 35 82  
19 56 13  
19 65 3  
19 66 74  
19 78 39  
19 80 17  
19 84 61  
19 93 99  
19 94 84  
19 96 9  
19 97 91  
19 99 40  
20 5 29  
20 14 55  
20 23 90  
20 31 72  
20 35 79  
20 36 43  
20 38 84  
20 43 7  
20 53 80  
20 55 71  
20 59 87  
20 62 47  
20 66 49  
20 73 12  
20 79 47  
20 81 88  
20 82 64  
20 83 4

20 94 93  
20 97 85  
20 99 63  
21 2 55  
21 4 65  
21 7 37  
21 13 7  
21 14 69  
21 15 33  
21 19 77  
21 20 70  
21 24 60  
21 34 47  
21 41 21  
21 50 32  
21 51 21  
21 57 71  
21 61 11  
21 65 69  
21 70 94  
21 71 91  
21 75 90  
21 76 39  
21 80 46  
21 92 95  
21 94 33  
22 4 76  
22 9 82  
22 19 36  
22 28 50  
22 30 23  
22 32 48  
22 33 53  
22 35 81  
22 36 43  
22 52 73  
22 69 46  
22 72 32  
22 75 29  
22 77 57  
22 80 36  
22 87 24  
22 88 42  
22 93 33  
23 1 58  
23 9 86  
23 10 61  
23 15 45  
23 17 99  
23 24 89  
23 25 24  
23 26 76  
23 42 40  
23 49 3  
23 50 41  
23 54 82  
23 55 46  
23 57 66

23 58 29  
23 59 74  
23 65 54  
23 67 84  
23 68 39  
23 69 35  
23 76 40  
23 77 47  
23 94 24  
23 98 43  
24 1 84  
24 2 77  
24 4 6  
24 13 60  
24 15 34  
24 17 16  
24 26 6  
24 30 12  
24 31 88  
24 34 32  
24 36 65  
24 42 63  
24 49 31  
24 51 71  
24 53 78  
24 55 61  
24 61 62  
24 65 56  
24 74 61  
24 75 62  
24 81 52  
24 83 74  
24 84 27  
24 92 26  
24 93 13  
25 2 87  
25 6 93  
25 14 11  
25 17 23  
25 19 66  
25 24 82  
25 29 75  
25 34 91  
25 39 21  
25 42 84  
25 47 38  
25 60 25  
25 62 5  
25 65 16  
25 66 59  
25 69 1  
25 71 4  
25 72 41  
25 75 9  
25 77 71  
25 78 60  
25 80 72  
25 83 52

25 96 42  
26 6 88  
26 9 57  
26 10 26  
26 13 32  
26 17 96  
26 19 41  
26 22 74  
26 36 53  
26 39 71  
26 41 51  
26 45 11  
26 46 96  
26 47 36  
26 53 36  
26 56 88  
26 58 77  
26 65 57  
26 71 37  
26 73 5  
26 81 17  
26 84 75  
26 90 60  
26 91 64  
27 1 96  
27 15 97  
27 16 93  
27 20 48  
27 23 80  
27 31 13  
27 38 73  
27 39 3  
27 43 62  
27 46 5  
27 51 98  
27 65 78  
27 68 39  
27 76 17  
27 86 93  
27 88 72  
27 92 25  
27 99 75  
28 4 31  
28 15 19  
28 22 11  
28 24 32  
28 25 31  
28 32 30  
28 33 97  
28 35 13  
28 38 54  
28 48 22  
28 50 31  
28 66 76  
28 78 80  
28 87 28  
28 92 1  
28 94 16

28 96 93  
28 98 47  
29 4 83  
29 6 56  
29 9 7  
29 12 80  
29 20 20  
29 21 79  
29 25 16  
29 26 61  
29 27 13  
29 35 33  
29 41 19  
29 43 39  
29 45 73  
29 47 2  
29 48 75  
29 52 52  
29 65 45  
29 73 86  
29 95 34  
30 0 78  
30 4 81  
30 6 41  
30 24 61  
30 28 26  
30 36 4  
30 55 20  
30 73 4  
30 76 40  
30 83 92  
30 92 64  
31 4 91  
31 5 55  
31 35 79  
31 37 92  
31 41 16  
31 43 74  
31 45 93  
31 54 58  
31 56 36  
31 57 9  
31 60 98  
31 62 90  
31 85 96  
31 88 13  
31 90 64  
32 0 56  
32 8 35  
32 9 44  
32 14 25  
32 27 74  
32 30 53  
32 31 49  
32 36 14  
32 53 53  
32 58 33  
32 60 47

32 62 67  
32 64 14  
32 66 3  
32 68 17  
32 82 60  
32 88 32  
32 89 3  
32 92 60  
32 95 58  
32 96 17  
33 4 79  
33 5 65  
33 7 18  
33 11 26  
33 19 51  
33 24 61  
33 27 62  
33 34 97  
33 46 81  
33 47 97  
33 51 17  
33 56 12  
33 58 77  
33 62 63  
33 65 85  
33 68 4  
33 71 26  
33 83 45  
33 84 47  
33 94 7  
34 0 84  
34 11 26  
34 19 13  
34 24 83  
34 30 84  
34 31 69  
34 32 90  
34 33 33  
34 45 28  
34 50 14  
34 53 93  
34 56 98  
34 60 75  
34 65 15  
34 68 73  
34 70 89  
34 74 30  
34 80 35  
34 86 65  
34 90 87  
34 94 35  
34 96 81  
34 99 60  
35 0 21  
35 6 45  
35 16 47  
35 17 48  
35 28 42

35 37 54  
35 42 66  
35 59 42  
35 70 65  
35 72 76  
35 77 76  
35 82 45  
35 83 65  
35 98 20  
36 17 28  
36 23 17  
36 24 5  
36 25 97  
36 26 77  
36 33 91  
36 38 36  
36 40 31  
36 54 44  
36 55 63  
36 56 92  
36 62 17  
36 63 36  
36 66 61  
36 70 59  
36 72 70  
36 75 45  
36 76 71  
36 84 76  
36 86 17  
36 95 36  
36 96 48  
36 99 16  
37 0 15  
37 5 35  
37 12 62  
37 13 67  
37 20 23  
37 22 11  
37 23 28  
37 28 31  
37 35 19  
37 39 9  
37 45 77  
37 55 65  
37 56 90  
37 59 75  
37 62 14  
37 70 3  
37 74 25  
37 80 53  
37 85 33  
37 94 31  
38 12 2  
38 15 22  
38 24 10  
38 42 26  
38 46 85  
38 47 9

38 49 85  
38 50 91  
38 54 3  
38 58 24  
38 59 43  
38 68 38  
38 69 94  
38 71 57  
38 72 85  
38 75 90  
38 76 90  
38 80 36  
38 84 17  
38 86 78  
38 87 70  
38 98 15  
39 3 99  
39 13 23  
39 16 96  
39 18 71  
39 24 69  
39 27 5  
39 29 64  
39 32 7  
39 33 13  
39 34 13  
39 35 30  
39 36 59  
39 43 75  
39 48 28  
39 55 48  
39 56 78  
39 61 69  
39 65 72  
39 67 59  
39 75 74  
39 77 42  
39 78 45  
39 79 64  
39 81 12  
39 84 93  
39 86 48  
39 91 31  
39 93 44  
39 97 22  
40 8 34  
40 10 44  
40 12 50  
40 16 45  
40 18 80  
40 20 39  
40 26 98  
40 27 81  
40 28 2  
40 51 24  
40 59 34  
40 62 85  
40 79 43

40 84 4  
40 87 20  
40 92 80  
40 93 2  
40 94 44  
40 95 70  
41 0 90  
41 2 31  
41 6 93  
41 7 29  
41 10 79  
41 17 67  
41 25 81  
41 46 40  
41 50 54  
41 52 64  
41 61 87  
41 73 85  
41 76 15  
41 85 14  
41 97 66  
42 2 14  
42 6 63  
42 11 58  
42 12 61  
42 13 23  
42 14 43  
42 15 26  
42 18 59  
42 22 83  
42 25 90  
42 28 36  
42 33 95  
42 36 41  
42 39 87  
42 41 53  
42 51 32  
42 57 68  
42 58 83  
42 59 40  
42 60 45  
42 61 31  
42 83 28  
42 85 98  
42 86 37  
42 87 52  
42 98 34  
43 1 71  
43 2 72  
43 5 16  
43 12 88  
43 16 85  
43 20 80  
43 21 93  
43 27 41  
43 31 90  
43 33 92  
43 51 22

43 62 76  
43 66 8  
43 79 22  
43 84 36  
43 86 81  
43 90 12  
43 95 59  
43 96 45  
43 98 17  
44 5 30  
44 6 5  
44 8 7  
44 10 29  
44 13 99  
44 15 78  
44 18 83  
44 25 68  
44 26 60  
44 28 22  
44 39 97  
44 46 35  
44 51 65  
44 52 2  
44 56 37  
44 61 73  
44 62 98  
44 66 93  
44 69 24  
44 72 75  
44 77 32  
44 80 9  
44 81 31  
44 86 93  
44 90 83  
45 2 6  
45 3 65  
45 6 14  
45 13 37  
45 14 59  
45 17 87  
45 24 65  
45 25 72  
45 26 76  
45 28 6  
45 39 28  
45 62 33  
45 74 75  
45 75 77  
45 76 37  
45 81 94  
45 83 82  
45 86 4  
45 90 72  
45 97 28  
46 2 29  
46 5 3  
46 6 7  
46 16 9

46 17 50  
46 18 84  
46 22 27  
46 25 56  
46 26 27  
46 30 22  
46 45 11  
46 47 98  
46 48 21  
46 49 7  
46 58 76  
46 59 63  
46 62 91  
46 65 74  
46 66 85  
46 79 93  
46 81 77  
46 88 53  
46 95 13  
47 2 6  
47 10 31  
47 13 1  
47 21 62  
47 31 75  
47 37 3  
47 41 45  
47 43 20  
47 57 58  
47 65 42  
47 67 53  
47 81 15  
47 90 91  
47 91 28  
47 92 88  
47 97 47  
48 2 31  
48 3 30  
48 6 71  
48 7 15  
48 8 47  
48 13 99  
48 17 68  
48 18 35  
48 21 10  
48 24 35  
48 29 78  
48 33 30  
48 35 22  
48 39 19  
48 40 40  
48 46 47  
48 59 70  
48 64 83  
48 74 41  
48 79 10  
48 84 20  
48 91 75  
49 4 39

49 5 61  
49 11 91  
49 14 70  
49 17 26  
49 18 80  
49 21 85  
49 22 20  
49 25 32  
49 29 75  
49 40 83  
49 41 22  
49 44 45  
49 47 94  
49 48 44  
49 50 43  
49 57 26  
49 59 21  
49 60 36  
49 61 61  
49 65 16  
49 66 75  
49 67 27  
49 70 47  
49 79 83  
49 84 59  
49 85 83  
49 86 75  
49 87 70  
49 95 72  
49 96 37  
49 97 24  
49 98 30  
49 99 51  
50 2 62  
50 3 24  
50 9 61  
50 16 16  
50 22 1  
50 36 64  
50 44 37  
50 57 19  
50 71 12  
50 79 1  
50 84 24  
50 88 70  
50 98 95  
50 99 83  
51 15 44  
51 30 31  
51 34 29  
51 36 85  
51 43 94  
51 44 25  
51 46 71  
51 54 35  
51 59 10  
51 65 83  
51 78 76

51 93 77  
51 94 51  
51 95 68  
52 1 40  
52 2 31  
52 4 10  
52 6 42  
52 7 75  
52 8 27  
52 9 69  
52 13 12  
52 16 91  
52 19 54  
52 25 72  
52 27 77  
52 28 4  
52 30 82  
52 38 4  
52 40 47  
52 46 28  
52 54 59  
52 70 57  
52 71 86  
52 73 27  
52 78 99  
52 79 22  
52 80 39  
52 82 86  
52 96 27  
53 2 4  
53 4 39  
53 9 76  
53 10 47  
53 12 2  
53 16 25  
53 23 93  
53 24 66  
53 30 48  
53 33 14  
53 41 80  
53 42 40  
53 45 39  
53 49 28  
53 50 17  
53 52 27  
53 60 41  
53 66 21  
53 68 97  
53 71 76  
53 75 30  
53 79 14  
53 82 44  
53 84 88  
53 85 20  
53 87 61  
54 2 98  
54 4 11  
54 7 20

54 15 23  
54 17 88  
54 18 60  
54 19 3  
54 21 33  
54 29 14  
54 34 50  
54 39 14  
54 40 19  
54 49 46  
54 50 61  
54 53 6  
54 63 32  
54 64 68  
54 65 94  
54 77 68  
54 79 46  
54 81 5  
54 83 14  
54 84 92  
54 87 90  
54 90 33  
54 93 12  
54 97 30  
55 3 58  
55 8 16  
55 22 8  
55 25 9  
55 33 14  
55 37 10  
55 39 1  
55 41 38  
55 52 28  
55 54 78  
55 56 61  
55 60 90  
55 61 89  
55 75 89  
55 76 49  
55 79 20  
55 84 73  
55 86 67  
56 2 49  
56 3 21  
56 4 12  
56 9 4  
56 14 8  
56 35 80  
56 37 26  
56 49 1  
56 52 16  
56 53 1  
56 57 87  
56 63 75  
56 70 95  
56 72 26  
56 74 6  
56 94 21

56 97 11  
56 98 51  
57 5 66  
57 7 98  
57 13 30  
57 20 95  
57 23 18  
57 27 49  
57 31 95  
57 32 95  
57 36 62  
57 44 75  
57 53 33  
57 55 65  
57 74 51  
57 81 59  
57 89 26  
57 95 47  
58 1 36  
58 4 38  
58 11 98  
58 12 41  
58 14 59  
58 21 45  
58 22 25  
58 23 17  
58 24 93  
58 27 18  
58 30 46  
58 33 18  
58 35 54  
58 36 91  
58 42 94  
58 45 19  
58 71 7  
58 75 74  
58 79 43  
58 85 87  
58 88 43  
58 89 3  
58 96 22  
59 3 77  
59 10 96  
59 17 56  
59 18 22  
59 20 15  
59 28 91  
59 29 85  
59 34 87  
59 36 71  
59 39 61  
59 42 62  
59 43 40  
59 50 81  
59 53 40  
59 62 13  
59 64 22  
59 67 68

59 78 66  
59 79 50  
59 84 79  
59 87 28  
59 91 29  
59 98 45  
60 0 68  
60 12 61  
60 15 2  
60 29 94  
60 32 8  
60 34 27  
60 38 41  
60 40 17  
60 43 79  
60 45 77  
60 53 60  
60 56 4  
60 62 17  
60 74 20  
60 77 11  
60 79 27  
60 86 65  
60 91 48  
60 94 80  
61 0 66  
61 3 79  
61 5 19  
61 15 7  
61 25 65  
61 27 49  
61 30 96  
61 32 46  
61 39 70  
61 48 13  
61 55 12  
61 56 3  
61 60 70  
61 70 86  
61 73 83  
61 80 62  
61 81 6  
61 84 64  
61 86 34  
61 88 37  
61 90 63  
61 96 49  
61 98 92  
62 2 97  
62 8 23  
62 11 97  
62 16 98  
62 39 31  
62 40 83  
62 64 58  
62 66 23  
62 67 5  
62 72 66

62 81 97  
62 82 93  
62 83 46  
62 85 79  
62 97 42  
63 0 88  
63 9 71  
63 17 16  
63 18 8  
63 20 85  
63 21 68  
63 22 81  
63 24 57  
63 27 68  
63 30 11  
63 40 63  
63 42 54  
63 44 57  
63 46 20  
63 52 94  
63 58 24  
63 64 91  
63 68 33  
63 71 92  
63 72 27  
63 73 59  
63 92 39  
63 97 65  
63 98 64  
63 99 62  
64 8 55  
64 22 94  
64 35 10  
64 40 68  
64 46 39  
64 49 1  
64 50 61  
64 54 68  
64 56 72  
64 62 50  
64 71 25  
64 73 86  
64 74 47  
64 79 75  
64 81 89  
64 83 25  
64 84 56  
64 92 15  
64 99 49  
65 9 2  
65 14 80  
65 26 95  
65 29 96  
65 34 67  
65 39 32  
65 40 25  
65 42 54  
65 48 94

```
65 49 56
65 50 97
65 51 95
65 52 18
65 55 37
65 59 51
65 64 38
65 69 46
65 70 97
65 80 51
65 88 31
65 95 5
66 3 7
66 4 40
66 5 79
66 7 82
66 13 38
66 18 93
66 32 38
66 33 43
66 46 99
66 54 68
66 58 28
66 63 36
66 65 17
66 76 30
66 87 10
66 92 64
66 96 31
67 0 98
67 2 87
67 3 92
67 7 12
67 26 77
67 28 11
67 29 52
67 36 1
67 46 38
67 49 70
67 52 73
67 56 31
67 59 5
67 62 42
67 63 93
67 66 89
67 77 5
==== end ====
ExitCode: 0
InstLimit: -1
*/
// input:
// n m
// u_1 v_1 w_1
// ...
// u_m v_m w_m
// (1 <= w_m <= 100)
```

```
int n;

int[][] a;

void init() {
 n = getInt();
 a = new int[n][n];
}

int main() {
 init();
 int i;
 int j;
 for (i = 0; i < n; ++i) {
 for (j = 0; j < n; ++j) {
 a[i][j] = 0;
 }
 }

 for (i = 98; i < n; ++i) {
 print("-1");
 }
 return 0;
}
```

```
/*
Test Package: Sema_Local_Preview
Test Target: Expression
Author: Pikachu
Time: 2019-10-20
Verdict: Success
Origin Package: Semantic Extended
*/
int main() {
 int a=1;
 printlnInt(+++a);
}
```

出bug | qwq

2.11

- default 没必要
- ++++a 寄
- 
- ```
/*
Test Package: Sema_Local_Preview
Test Target: Misc
Author: 15' Zhekai Zhang
Time: 2019-10-20
Verdict: Success
Origin Package: Semantic Extended
*/
int countA;
```

```

int countB;
int countC;
C something;

class A
{
    A a;
    B b;
    A()
    {
        idx = countA++;
        if(idx % 2 == 0)
        {
            a = new A;
            if(countB % 2 == 0)
                b = new B;
            else
                b = null;
        }
        else
            a = null;
        c = new C[2][];
        c[0] = (new C[6][6][6][6])[2][3][3];
        c[1] = null;
        if(c.size() != 2)
            println("oops!");
    }
    C[] getc0()
    {
        return c[0];
    }
    C[][] c;
    int idx;
};

class B
{
    int idx;
    C c;
    B()
    {
        idx = countB++;
        c = (new A).getc0()[0].Me().Me();
    }
};

class C
{
    int idx;
    C()
    {
        this.idx = countC++;
        str = toString(idx);
        something = Me();
    }
    C Me() { return this; }
    string str;
};

```

```

void count()
{
    countA = 0;
    countB = 0;
    countC = 0;
    B b = new B;
    println(toString(countA) + " " + toString(countB) + " " +
toString(countC));
    countA = 1;
    countB = 1;
    countC = 1;
    b = new B;
    print(toString(countA-1) + " " + toString(countB-1) + " " +
toString(countC-1));
    print("\n");
    println(toString(something.Me().str.substring(1,
something.str.length()-1).parseInt()));
    string temp = toString(something.str.ord(42 & 21));
    if(temp < something.str)
        println(something.str + ">" + temp);
    else
        println(something.str + "<=" + temp);
}

///////////////////////////////
int main()
{
    {{};{};{}};
    int i;
    for(i=0;;i++)
        if(((i ^ 891 & 759) == 666) == !false)
    {
        println(tostring(i));
        int i = 0;
        println(toString(i));
        {
            int i = 1;
            println(toString(i));
        }
        count();
        break;
    }
    while(true)
    {
        if(i % 2 == 0)
            continue;
        print(toString(i)+",");
    }
    println("");
    return 0;
}

```

2.13优化print

/*

- optimize print
 -
 - print(A + B + C) -> print(A), print(B), print(C)
 - print(toString(A)) -> printInt(A)
 - */

Function Inlining

- Highly Effective! up to ~ 40% improvement
- Can be extended to recursion unrolling.

	hashmap	horse	horse3
Limit	550000	25000000	25000000
Original	248263	11911209	14027651
Original %	45.14%	47.64%	56.11%
Optimized	171953	3914219	4642421
Optimized %	31.26%	15.66%	18.57%

```
//print problem
```

```
/*
```

```
Test Package: Codegen
```

```
Author: 16' Kaiyi Zhang
```

```
Time: 2020-02-03
```

```
Input:
```

```
==== input ===
```

```
==== end ===
```

```
Output:
```

```
==== output ===
```

```
string digit(int x){
```

```
if(x==0) return "0";
```

```
if(x==1) return "1";
```

```
if(x==2) return "2";
```

```
if(x==3) return "3";
```

```
if(x==4) return "4";
```

```
if(x==5) return "5";
```

```
if(x==6) return "6";
```

```
if(x==7) return "7";
```

```
if(x==8) return "8";
```

```
if(x==9) return "9";
```

```
}
```

```
string[] s=new string[256];
```

```
string[] c=new string[256];
```

```
string s2(int ss){
```

```
if(ss<=9) return "s["+digit(ss)+"]=";
```

```
return "s["+digit(ss/10)+digit(ss%10)+"]=";
```

```
}
```

```
string c2(int cc){
if(cc<=9) return "c["+digit(cc)+"]=";
return "c["+digit(cc/10)+digit(cc%10)+"]=";
}
string co=";";
string a2q="\\";;
string a2b="\\";
int main(){int i=0;// Quine is a program that produces its source code as
output.
c[0]=" ";
c[1]!="";
c[2]="#";
c[3]="$";
c[4]= "%";
c[5]("&";
c[6]="";
c[7](";
c[8])";
c[9]("*";
c[10]+";
c[11]=",";
c[12]=-";
c[13]= ".";
c[14]/";
c[15]="0";
c[16]="1";
c[17]="2";
c[18]="3";
c[19]="4";
c[20]="5";
c[21]="6";
c[22]="7";
c[23]="8";
c[24]="9";
c[25]=":";
c[26]=";";
c[27]<"";
c[28]=";
c[29]>"";
c[30]=?";
c[31]="@";
c[32]="A";
c[33]="B";
c[34]="C";
c[35]="D";
c[36]="E";
c[37]="F";
c[38]="G";
c[39]="H";
c[40]="I";
c[41]="J";
c[42]="K";
c[43]="L";
c[44]="M";
c[45]="N";
c[46]="O";
c[47]="P";
c[48]="Q";
```

```
c[49]="R";
c[50]="S";
c[51]="T";
c[52]="U";
c[53]="V";
c[54]="W";
c[55]="X";
c[56]="Y";
c[57]="Z";
c[58]="[";
c[59]="]";
c[60]="^";
c[61]="_";
c[62]`\`;
c[63]="a";
c[64]="b";
c[65]="c";
c[66]="d";
c[67]="e";
c[68]="f";
c[69]="g";
c[70]="h";
c[71]="i";
c[72]="j";
c[73]="k";
c[74]="l";
c[75]="m";
c[76]="n";
c[77]="o";
c[78]="p";
c[79]="q";
c[80]="r";
c[81]="s";
c[82]="t";
c[83]="u";
c[84]="v";
c[85]="w";
c[86]="x";
c[87]="y";
c[88]="z";
c[89]={"";
c[90]="|";
c[91]}";
c[92]=~";
s[0]="int main(){int i=0;// Quine is a program that produces its source code as
output.";
s[1]="println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[66]+c[71]+c[69]+c[82]+c
[7]+c[71]+c[76]+c[82]+c[0]+c[86]+c[8]+c[89]);";
s[2]="println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[15]+c[8]+c[80]+c[67]+c[82]+c
[83]+c[80]+c[76]+c[0]+a2q+c[15]+a2q+c[26]);";
s[3]="println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[16]+c[8]+c[80]+c[67]+c[82]+c
[83]+c[80]+c[76]+c[0]+a2q+c[16]+a2q+c[26]);";
s[4]="println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[17]+c[8]+c[80]+c[67]+c[82]+c
[83]+c[80]+c[76]+c[0]+a2q+c[17]+a2q+c[26]);";
s[5]="println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[18]+c[8]+c[80]+c[67]+c[82]+c
[83]+c[80]+c[76]+c[0]+a2q+c[18]+a2q+c[26]);";
s[6]="println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[19]+c[8]+c[80]+c[67]+c[82]+c
[83]+c[80]+c[76]+c[0]+a2q+c[19]+a2q+c[26]);";
```

```

s[7] = "println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[20]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[20]+a2q+c[26]);";
s[8] = "println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[21]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[21]+a2q+c[26]);";
s[9] = "println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[22]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[22]+a2q+c[26]);";
s[10] = "println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[23]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[23]+a2q+c[26]);";
s[11] = "println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[24]+a2q+c[26]);";
s[12] = "println(c[91]);";
s[13] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[59]+c[0]+c[81]+c[28]+c[76]+c[67]+c[85]+c[0]+c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[17]+c[20]+c[21]+c[59]+c[26]);";
s[14] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[59]+c[0]+c[65]+c[28]+c[76]+c[67]+c[85]+c[0]+c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[17]+c[20]+c[21]+c[59]+c[26]);";
s[15] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[81]+c[17]+c[7]+c[71]+c[76]+c[82]+c[0]+c[81]+c[8];";
s[16] = "println(c[71]+c[68]+c[7]+c[81]+c[81]+c[27]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[81]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[7]+c[81]+c[81]+c[10]+a2q+c[59]+c[28]+a2q+c[26]);";
s[17] = "println(c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[81]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[71]+c[69]+c[82]+c[7]+c[81]+c[81]+c[4]+c[16]+c[15]+c[8]+c[10]+c[66]+c[71]+c[69]+c[82]+c[26]);";
s[18] = "println(c[91]);";
s[19] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[65]+c[17]+c[7]+c[71]+c[76]+c[82]+c[0]+c[65]+c[65]+c[8]+c[89]);";
s[20] = "println(c[71]+c[68]+c[7]+c[65]+c[65]+c[27]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[65]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[7]+c[65]+c[65]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26]);";
s[21] = "println(c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[65]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[71]+c[69]+c[82]+c[7]+c[65]+c[65]+c[4]+c[16]+c[15]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26]);";
s[22] = "println(c[91]);";
s[23] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[65]+c[77]+c[28]+a2q+c[26]+a2q+c[26]);";
s[24] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[63]+c[17]+c[79]+c[28]+a2q+a2b+a2q+a2q+c[26]);";
s[25] = "println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[63]+c[17]+c[64]+c[28]+a2q+a2b+a2b+a2q+c[26]);";
s[26] = "println(s[0]);";
s[27] = "for(i=0;i<93;i++)println(c2(i)+a2q+c[i]+a2q+co);";
s[28] = "for(i=0;i<32;i++)println(s2(i)+a2q+s[i]+a2q+co);";
s[29] = "for(i=1;i<32;i++)println(s[i]);";
s[30] = "return 0;";
s[31] = "}";
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[66]+c[71]+c[69]+c[82]+c[7]+c[71]+c[76]+c[82]+c[0]+c[86]+c[8]+c[89]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[15]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[15]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[16]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[16]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[17]+c[8]+c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[17]+a2q+c[26]);

```

```

println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[18]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[18]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[19]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[19]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[20]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[20]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[21]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[21]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[22]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[22]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[23]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[23]+a2q+c[26]);
println(c[71]+c[68]+c[7]+c[86]+c[28]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[83]+c[
80]+c[76]+c[0]+a2q+c[24]+a2q+c[26]);
println(c[91]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[59]+c[0]+c[81]+c[28]+c[76]+c[
67]+c[85]+c[0]+c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[17]+c[20]+c[21]+c[59]
]+c[26]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[59]+c[0]+c[65]+c[28]+c[76]+c[
67]+c[85]+c[0]+c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[58]+c[17]+c[20]+c[21]+c[59]
]+c[26]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[81]+c[17]+c[7]+c[71]+c[76]+c[
82]+c[0]+c[81]+c[81]+c[8]+c[89]);
println(c[71]+c[68]+c[7]+c[81]+c[81]+c[27]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[
83]+c[80]+c[76]+c[0]+a2q+c[81]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[7]+c[81]
]+c[81]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26]);
println(c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[81]+c[58]+a2q+c[10]+c[66]
+c[71]+c[69]+c[82]+c[7]+c[81]+c[14]+c[16]+c[15]+c[8]+c[10]+c[66]+c[71]+c[6
9]+c[82]+c[7]+c[81]+c[81]+c[4]+c[16]+c[15]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26])
;
println(c[91]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[65]+c[17]+c[7]+c[71]+c[76]+c[
82]+c[0]+c[65]+c[65]+c[8]+c[89]);
println(c[71]+c[68]+c[7]+c[65]+c[65]+c[27]+c[28]+c[24]+c[8]+c[80]+c[67]+c[82]+c[
83]+c[80]+c[76]+c[0]+a2q+c[65]+c[58]+a2q+c[10]+c[66]+c[71]+c[69]+c[82]+c[7]+c[65]
]+c[65]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26]);
println(c[80]+c[67]+c[82]+c[83]+c[80]+c[76]+c[0]+a2q+c[65]+c[58]+a2q+c[10]+c[66]
+c[71]+c[69]+c[82]+c[7]+c[65]+c[65]+c[14]+c[16]+c[15]+c[8]+c[10]+c[66]+c[71]+c[6
9]+c[82]+c[7]+c[65]+c[65]+c[4]+c[16]+c[15]+c[8]+c[10]+a2q+c[59]+c[28]+a2q+c[26])
;
println(c[91]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[65]+c[77]+c[28]+a2q+c[26]+a2q
+c[26]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[63]+c[17]+c[79]+c[28]+a2q+a2b
+a2q+a2q+c[26]);
println(c[81]+c[82]+c[80]+c[71]+c[76]+c[69]+c[0]+c[63]+c[17]+c[64]+c[28]+a2q+a2b
+a2b+a2q+c[26]);
println(s[0]);
for(i=0;i<93;i++)println(c2(i)+a2q+c[i]+a2q+co);
for(i=0;i<32;i++)println(s2(i)+a2q+s[i]+a2q+co);
for(i=1;i<32;i++)println(s[i]);
return 0;
}
== end ==
ExitCode: 0
InstLimit: -1
Origin Package: Codegen Pretest-925
*/

```

```
string co="";
string a2q="\"";
string a2b="\\";
int main(){ int i=0;// Quine is a program that produces its source code as
output.

print("%");
print(a2q);
println("%"+a2q);

return 0;
}
```

SSA

4.4 learn by libro

先建立dominnate tree <https://blog.csdn.net/dashuniuniu/article/details/52224882>

然后虎书上面学习如何插入phi

这些统称为mem2reg 可以在mem2reg 第一篇博客上面找到 llvm 源码的解析

之后的所有优化我猜都是在DT上面进行

(三)自学笔记 其他知识 (阶段笔记)

一.正则表达式

R={“0”, “1”}

R* = {空串, “0”, “1”, “00”, “01” ...} 表达所有有限二进制串。

1.简介

data(\w)?.dat

```
data.dat
data1.dat
data2.dat
datax.dat
dataN.dat
```

- [0-9]+匹配多个数字, [0-9] 匹配单个数字, + 匹配一个或者多个。

^ 是匹配开始的地方 \$ 是匹配结束的地方

开始标记

3~15个字符的长度

`^[a-zA-Z0-9_]{3,15}$`

结束标记

字母(a-z)数字(0-9)下划线_连字符

• - 除换行符以外的所有字符。

^ - 字符串开头。

\$ - 字符串结尾。

\d, \w, \s - 匹配数字、字符、空格。

\D, \W, \S - 匹配非数字、非字符、非空格。

[abc] - 匹配 a、b 或 c 中的一个字母。

[a-z] - 匹配 a 到 z 中的一个字母。

[^abc] - 匹配除了 a、b 或 c 中的其他字母。

aa|bb - 匹配 aa 或 bb。

? - 0 次或 1 次匹配。

* - 匹配 0 次或多次。

+ - 匹配 1 次或多次。

一些语法

正则表达式也可以用于文本替换 匹配出一些单元并将其用特定的新的字符串替换

2.语法

- **runoo+b**, 可以匹配 runoob、runooob、runooooob 等，+ 号代表前面的字符必须至少出现一次 (1次或多次) 。
- **runoo*b**, 可以匹配 runob、runoob、runooooob 等，* 号代表前面的字符可以不出现，也可以出现一次或者多次 (0次、或1次、或多次) 。
- **colou?r** 可以匹配 color 或者 colour，? 问号代表前面的字符最多只可以出现一次 (0次、或1次) 。

经典的匹配模式

\w 比较常用 匹配字母、数字、下划线。等价于 [A-Za-z0-9_]

普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。这包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

字符	描述
[ABC]	<p>匹配 [...] 中的所有字符，例如 [aeiou] 匹配字符串 "google runoob taobao" 中所有的 e o u a 字母。</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content;"> <p><code>/[aeiou]/g</code></p> <p>Text Tests</p> </div>
[^ABC]	<p>匹配除了 [...] 中字符的所有字符，例如 [^aeiou] 匹配字符串 "google runoob taobao" 中除了 e o u a 字母的所有字母。</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content;"> <p><code>/[^aeiou]/g</code></p> <p>Text Tests</p> </div>
[A-Z]	<p>[A-Z] 表示一个区间，匹配所有大写字母，[a-z] 表示所有小写字母。</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content;"> <p><code>/[A-Z]/g</code></p> <p>Text Tests</p> </div>
.	<p>匹配除换行符 (\n、\r) 之外的任何单个字符，相等于 [^\n\r]。</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content;"> <p><code>/./g</code></p> <p>Text Tests</p> </div>
[\s\S]	<p>匹配所有。 \s 是匹配所有空白符，包括换行，\S 非空白符，不包括换行。</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content;"> <p><code>/[\s\S]/g</code></p> <p>Text Tests</p> </div>

限定符

限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。有 * 或 + 或 ? 或 {n} 或 {n,} 或 {n,m} 共6种。

正则表达式的限定符有：

字符	描述
*	匹配前面的子表达式零次或多次。例如，'zo*' 能匹配 "z" 以及 "zoo"。* 等价于 {0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do"、"does" 中的 "does"、"doxy" 中的 "do"。? 等价于 {0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooooo" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中 n <= m。最少匹配 n 次且最多匹配 m 次。例如，'o{1,3}' 将匹配 "foooooo" 中的前三 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

以下正则表达式匹配一个正整数，[1-9]设置第一个数字不是 0，[0-9]* 表示任意多个数字：

```
/[1-9][0-9]*/
```

关于贪婪匹配和非贪婪匹配

* 和 + 限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个 ? 就可以实现非贪婪或最小匹配。

通过在 *、+ 或 ? 限定符之后放置 ?，该表达式从“贪婪”表达式转换为“非贪婪”表达式或者最小匹配。

\d 匹配一个数字字符。等价于 [0-9]。

. 代表任意字符

单词匹配的边界问题

\b	匹配一个单词边界，也就是指单词和空格间的位置。例如，'er\b' 可以匹配 "never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。

字符集

[] 来表示

```
\b[\w.%+-]+@[ \w.-]+\.[a-zA-Z]{2,6}\b
```

[] 内部的 . 就是. 没有其他含义 上面这个正则表达式能匹配邮箱

二.antlr学习

1. 博客汇总

lexer是对语句进行词法分析，切分为不同的token

use the generated lexer and parser: you invoke them passing the code to recognize and they return to you a parse tree

AST的作用

You create the AST by manipulating the parse tree, in order to get something that is easier to use by subsequent parts of your program.

1. 文法规则和词法规则可以同时存在一个文件中，但文（语）法（语法规则rule）以小写开头，词法以大写开头(词法规则token)。

输入流`先经过`词法分析器`生成匹配的`词法符号流`，词法符号流经过`语法分析器`生成匹配的`语法结构

词法分析 (Lexical analysis或Scanning) 和词法分析程序 (Lexical analyzer或Scanner) 词法分析阶段是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符地读入源程序，即对构成源程序的字符流进行扫描然后根据构词规则识别单词(也称单词符号或符号)。

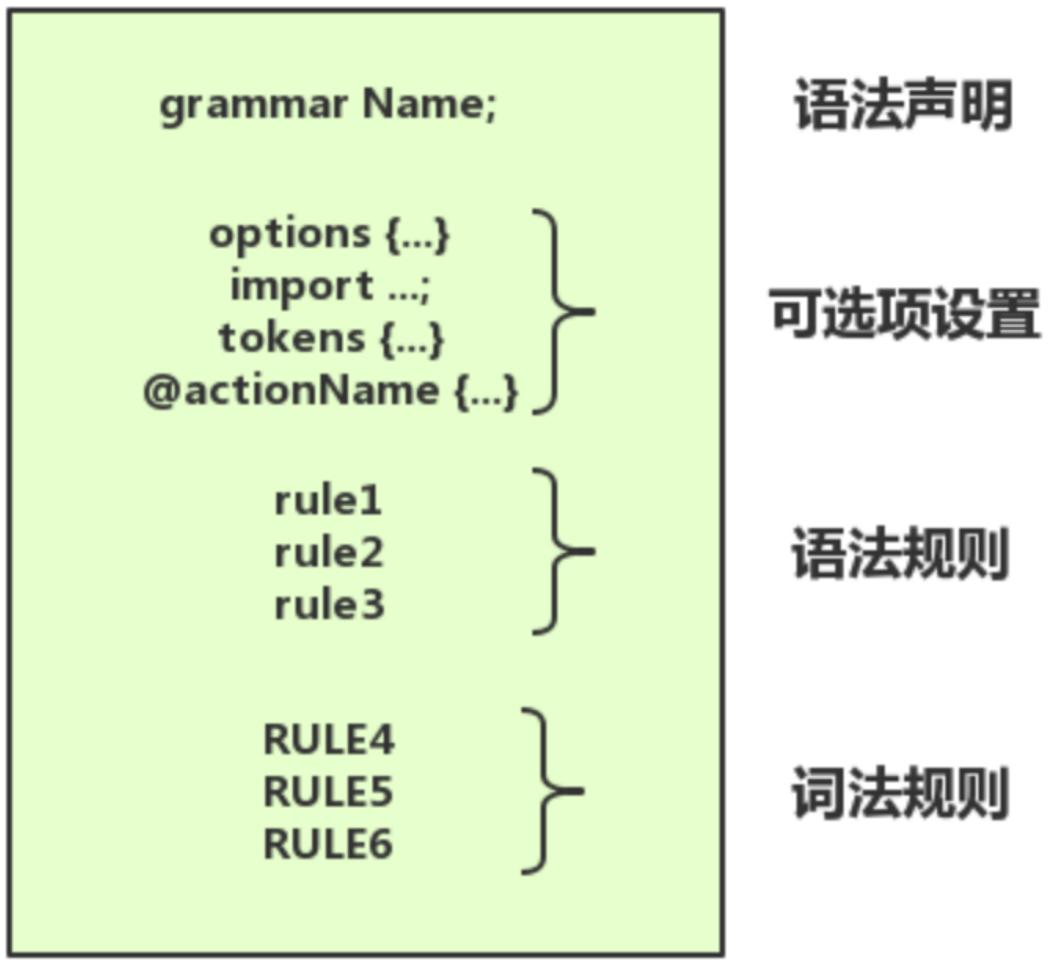
语法分析 (Syntax analysis或Parsing) 和语法分析程序 (Parser) 语法分析是编译过程的一个逻辑阶段。语法分析的任务是在词法分析的基础上将单词序列组合成各类语法短语，如“程序”，“语句”，“表达式”等等

最后只有语法会出现在语法树中间

// /**/作为注释

#作为提示符号

语法分析器的工作可以分解为两个步骤：第一步是词法分析 (lexical analysis)，读取字符流，将字符聚集为单词或者符号 (Token)，并标记每一个符号的类别。比如一条赋值语句 `a = 20;`，词法分析之后的结果为 `a`, `=`, `20`, `;` 四个符号，并标记 `a` 为ID, `20` 为INT数值等。第二步是语法分析 (syntax analysis)，语法分析通过解析词法分析生成的符号来识别语句结构，它一般不在乎符号的内容，只关心它们的类型。上边的语句经过语法分析后，就会得出这是 赋值语句 的结论。



{xxx}.g4

g4架构

2.terr的书

设计的起始规则就是使用英语伪代码来描述输入文本的整体结构

一个可行的分解：文件-行-字段（field）

比如java

规则名）。在最粗的粒度上，一个Java的编译单元（compilation unit）由一个可选的包声明语句（package specifier）和一个或多个类定义（class definition）组成。其中，类定义由关键字class开始，之后是一个标识符、可选的父类名（superclass specifier）、可选的实现语句（implements clause），以及类的定义体（class body）。一个类的定义体就是由花括号包裹的一系列类成员（class member）。一个类成员可以是内部类定义、字段或者方法。然后，我们将会描述字段和方

[1].语法

(1).使用antlr 识别常见的语言模式

序列模式

选择模式 (多个备选的分支)

词法符号依赖模式 () [] {} 成对出现

嵌套模式 自相似的语言结构

如下图 (类似递归)

```
expr: ID '[' expr ']' // a[1], a[b[1]], a[(2*b[1])]
      | '(' expr ')'   // (1), (a[1]), (((1))), (2*a[1])
      | INT             // 1, 94117
      ;
```

(2).处理优先级，左递归和结合性

优先级高的写在前面

```
expr : expr '^'<assoc=right> expr // ^ 运算符是右结合的
      | INT
      :
```

注：在ANTLR 4.2之后，`<assoc=right>`需要被放到备选分支的最左侧，否则会收到警告。在本例中，正确写法是：

```
expr : <assoc=right> expr '^' expr
      | INT
      ;
```

处理右结合

=是右结合的，所以先计算(b+c+d)，然后再赋值给a

+是左结合的，所以先计算(b+c)，然后再计算(b+c)+d

C语言中具有右结合性的运算符包括所有单目运算符 (~取反、!取非、-负号、+正号、++自增、-自减) 以及赋值运算符 (=) 和条件运算符 (?:)。其它运算符都是左结合性。

[2].词法

基本的标点符号 关键词 标识符 数字 字符串的写法

Vistor/Listener

- 监听器的方法会被ANTLR提供的遍历器对象自动调用，即当进入和离开对应节点时会调用对应方法。而访问器必须显式调用visit方法来访问子节点，否则子节点将不会被访问。
- ANTLR会生成Vistor和Listener的接口 `xxVisiter` 和一个默认实现 `xxBaseListener`，我们可以选择继承 `xxBaseListener` 并覆盖我们需要修改的方法。

Parser

- `ProgramContext`:
 - 用 `MxParser.program()` 可接到 `ProgramContext` (Derived from parse tree)
 - `public List<SubProgramContext> subProgram()` 返回子节点形成的 `ArrayList<Sub...>`
源于 `program: subProgram *`
 - 覆写了 `public <T> T accept(ParseTreeVisitor<? extends T> visitor)` 函数
在 `AbstractParseTreeVistor` 实现的 `visit` 函数 即 `ParseTree.accept(this)` 实际上调用了该 `accept` 函数
而该 `accept` 函数的实现又调用了 `ParseTreeVisitor.visitProgram` 也即实际调用 `ASTBuilder` 中覆写的函数
 - => `visit(Type ctx)` 调用 `ASTBuilder.visitType(ctx)` 返回一个自定义的 `ASTNode`

- `SubProgramContext`

```
subProgram
:   functionDecl    #functionDeclaration
|   classDecl       #classDeclaration
|   variableDecl   #variableDeclaration
;
```

- 实际上是作为 父类 发挥作用 提高代码复用率
`FunctionDeclarationContext`、`ClassDeclarationContext`、`VariableDeclarationContext` 均 extends `subProgramCtx`
- 其子类均实现了 `accept` 函数(等价于 `ASTBuilder.visit(子类ctx)` 返回子类节点)
由于向上转型 传一个父类的引用进入函数当中是可以调用其 `accept` 函数的 (由此实现多态)
- Usage:

```
ArrayList<ASTNode(or other father node)> _List;
ProgramCtx.subProgram.forEach(ptr->_List.add((ASTNode) visit(ptr)));
```

- `FunctionDeclarationContext`、`ClassDeclarationContext`、`VariableDeclarationContext`
 - 类似于 `ProgramContext` 覆写 `accept` 需要实现 `visitxxx(xxx ctx)`;
 - 其成员函数即为返回对应的 `functionDeclCtx`、`classDeclCtx`、`variableDeclCtx`
 - ! 这样打标签实在是多此一举! (进行一个 modify)
 - 打标签的实质是: 告诉ANTLR不要为 `subProgram` 生成实际的 `accept` 而是把其当作父类, 其各种组件打上标签及都要生成对应的 `ParseRuleContext && accept && extends subProgram`。也即这个文法实际上由其他文法定义而成, 只不过分类在一起。

```
//After modification
subProgram
:   functionDecl
|   classDecl
|   variableDecl
;
public static class SubProgramContext extends ParserRuleContext {
    public FunctionDeclContext functionDecl();
```

```

    public ClassDeclContext classDecl();
    public VariableDeclContext variableDecl();
    public SubProgramContext(ParserRuleContext parent, int
invokingState);
        @Override public <T> T accept(ParseTreeVisitor<? extends T>
visitor);
    }
    //既有accept 又能向下到子节点 实为文法

```

■ **Necessity** for tag Expressions :

给每一种分支都覆写一个 accept 函数，而 expression 本身当作父类起到多态作用。

即abstract class ExprNode: BinaryExprNode/NoaryExprNode extends ExprNode.

arraytype只在 builder visitarraytype中被访问过，且他不断递归调用自身直达到最底层的basetype

三.AST 构建

1. 博客汇总

abstract syntax tree 源代码的抽象语法结构，树上的每个节点都代表源代码中的一个结构

核心是生成节点 继承自bode类来表示单个节点，继承出的类很多 层次也很复杂，先考虑重要的一级类

类名	作用
AST	表示抽象语法树的根的节点类
StmtNode	表示语句的节点的基类
ExprNode	表示表达式的节点的基类
TypeDefinition	定义类型的节点的基类

重要的节点类

node有location属性 报错信息存储在location里面

- 表达式的ast
 - 字面量的ast
 - 一元二元运算ast
 - 条件表达式ast
 - 赋值表达式ast
- 语句ast
 - if
 - while
 - 程序块
- 声明ast
 - 变量声明 函数定义
 - 表示函数整体

抽象语法树是一个深度嵌套的对象，这个对象以一种既能够简单地操作又提供很多关于源代码信息的形式，来展现场码。

ast元模型看起来和parser tree元模型相当类似 即antlr商城的包含节点的类集

一些区别：

- 比parser tree有更简单的api
- 可以删除一些解析时有用但是在逻辑上没有什么用的元素（比如括号 行节点）这也是抽象语法树抽象的原因 抽象语法树并不会呈现出真实语法的每一个细节 嵌套括号被隐藏在树的结构中 并没有以节点的形式呈现出来

抽象语法树 是编译器中广泛使用的表示程序代码结构的数据结构。AST 通常是编译器 语法分析 阶段的结果。

四.ir学习

(1) [北航教程][https://buaa-se-compiling.github.io/minisysY-tutorial/pre/llvm_ir_quick_primer.html]

1.快速入门

```
clang -S -emit-llvm test.c
```

生成中间代码

```
clang -S -emit-llvm -O3 test.c
```

O3优化

```
llc test.ll
```

生成汇编代码

然后用操作系统自带的汇编器和链接器生成可执行文件

这是一个基于llvm的编译器

```
.c --frontend--> AST --frontend--> LLVM IR --LLVM opt--> LLVM IR --LLVM llc--> .s  
Assembly --OS Assembler--> .o --OS Linker--> executable
```

手册<https://llvm.org/docs/LangRef.html>

- 关于llvm 和clang的关系

Clang 作为 LLVM project 的一个子项目，是 LLVM 项目中 c/c++/obj-c 语言的前端，其用法与 GCC 基本相同

用ir的好处：

1.有一些优化技术是目标平台无关的（例如作为我们实验挑战任务的死代码删除和常量折叠），我们只需要在 IR 上做这些优化，再翻译到不同的汇编，可以减少工作量

2.减少编译器的数量

前端：源语言变成ir

中端:ir的优化

后端：生成ricsv(编译成目标语言)

llvm有三种表达形式 其中.ll是人可读的代码语言（其他形式的也是等价的）

- 读llvm

`align` 字段描述了程序的对齐属性; `dso_local` 是变量和函数的运行时抢占说明符; 以`;`开头的字符串是 LLVM IR 的注释 (这个最好记住)

`alloc` 是指针

局部变量的作用域是单个函数

全局变量与局部变量由前缀区分, 全局变量和函数名以`@`为前缀, 局部变量以`%`为前缀

内联函数: 作为代码直接插入省去了跳转的代价 是优化的一个方面

- `br` 指令

```
br + 标志位 + trueLabel + falseLabel` , 或者 `br + label` (强制跳转)
```

- `icmp`

```
%9 = icmp eq i32 %7, %8
```

相等置为1 不相等置为0

类型系统 LLVM 的优秀之处

integer type `i1` `i32`

label type

后面还有数组 type 和变量 type

2.ssa

存取内存形式的 IR 以及带有 `phi` 指令的 SSA IR

静态单赋值 (Static Single Assignment, SSA)

且每个变量只能被赋值一次 (如果套用 C++ 的术语, 就是说每个变量只能被初始化, 不能被赋值) 类似 rust...?

- 对于 for 循环 load/store 形式可以改为 Phi 形式

```
3: ; preds = %5, %1
  %4 = phi i32 [ 1, %1 ], [ %8, %5 ] ; 如果是从块 %1 来的, 那么值就是
1, 如果是从                                ; 块 %5 来的, 那么值就是 %8 的值
  ret i32 %4
```

内存四字节对齐

```
align 4
```

```
// 把内存内容 load 进入寄存器
%6 = load i32, i32* %4, align 4
```

`alloca` 指令的作用是在当前执行的函数的栈帧上分配内存并返回一个指向这片内存的指针, 当函数返回时内存会被自动释放 (一般是改变栈指针)。

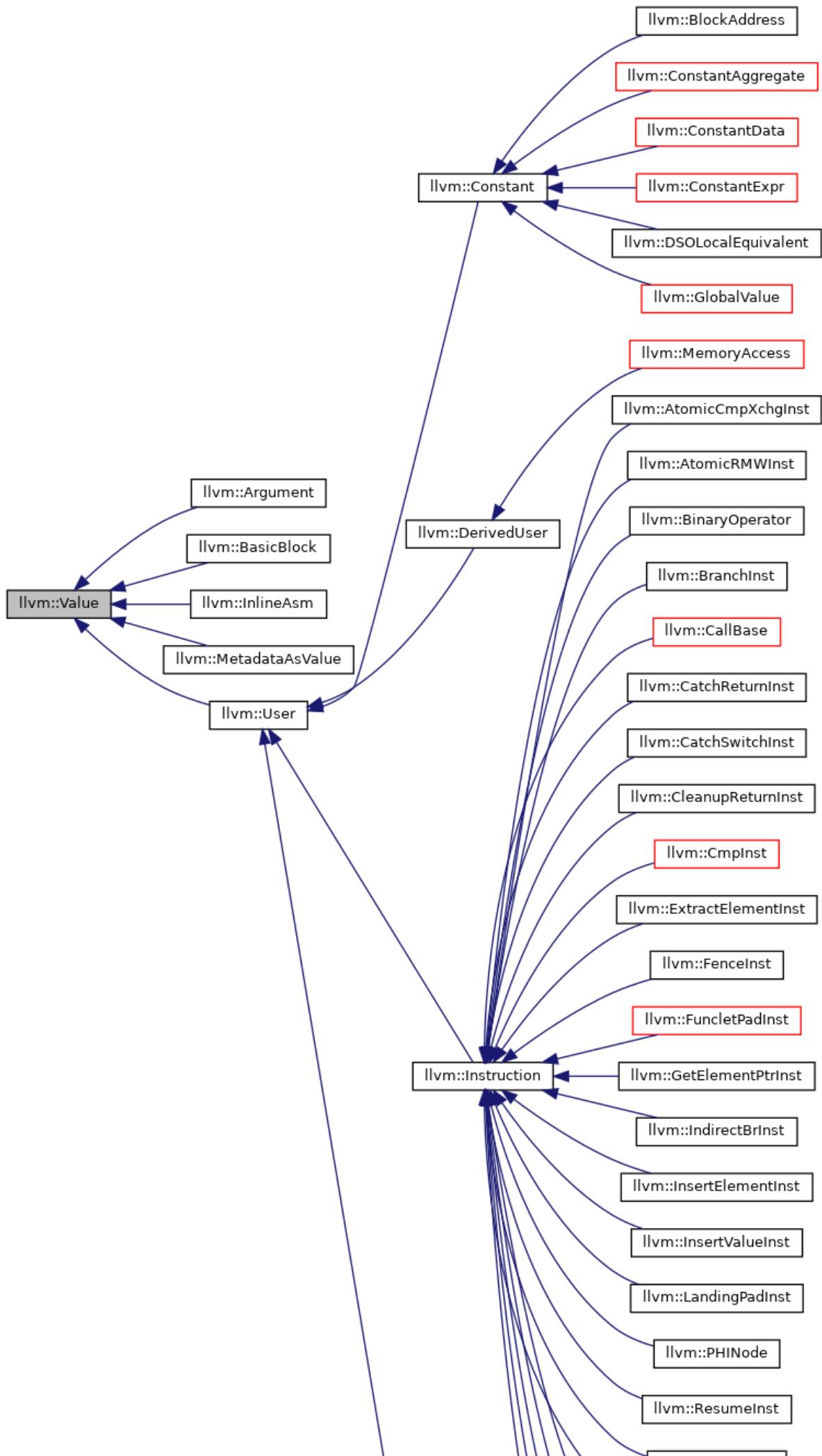
由于内存不需要 SSA, 可以经受多次数值的改变 这就是 load store 操作-

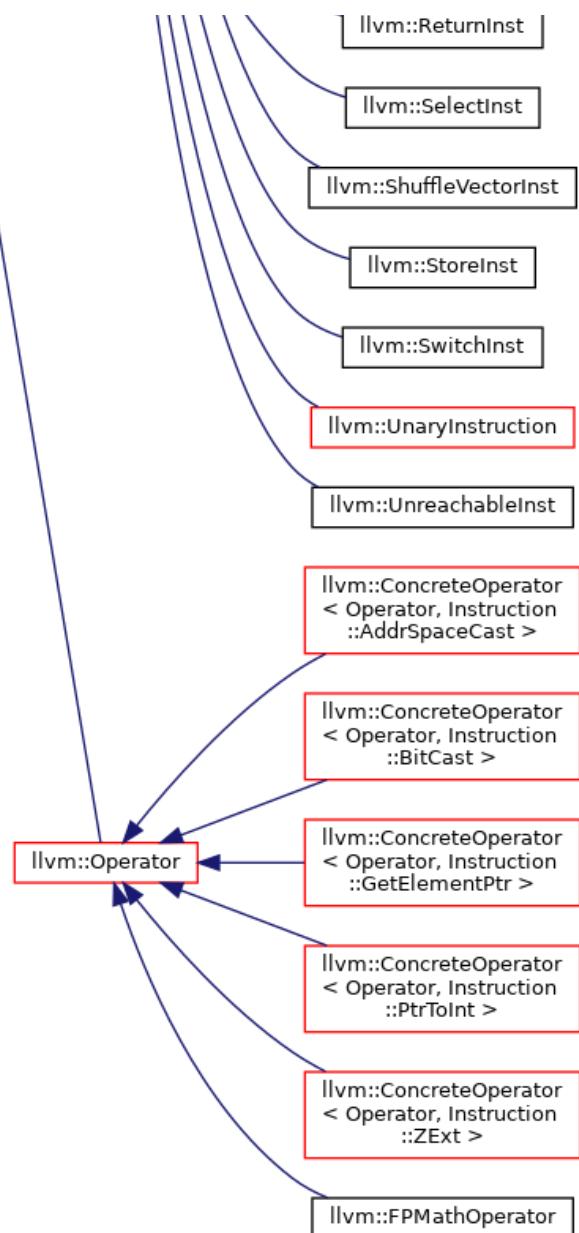
- 论 `alloca + mem2reg` 技术

alloca 在内存上可以无限次更改，但是频繁的内存访问会导致性能较差

mem2reg 对于每一个寄存器只能够一次赋值 要求比较高

3.value use user





instructions

llvm ir	usage	intro
add	<code><result> = add <ty> <op1>, <op2></code>	/
sub	<code><result> = sub <ty> <op1>, <op2></code>	/
mul	<code><result> = mul <ty> <op1>, <op2></code>	/
sdiv	<code><result> = sdiv <ty> <op1>, <op2></code>	有符号除法
icmp	<code><result> = icmp <cond> <ty> <op1>, <op2></code>	比较指令
and	<code><result> = and <ty> <op1>, <op2></code>	与
or	<code><result> = or <ty> <op1>, <op2></code>	或
call	<code><result> = call [ret attrs] <ty> <fnptrval>(<function args>)</code>	函数调用
alloca	<code><result> = alloca <type></code>	分配内存
load	<code><result> = load <ty>, <ty>* <pointer></code>	读取内存
store	<code>store <ty> <value>, <ty>* <pointer></code>	写内存
getelementptr	<code><result> = getelementptr <ty>, * {, [inrange] <ty> <idx>}* <result> = getelementptr inbounds <ty>, <ty>* <ptrval>{, [inrange] <ty> <idx>}*</code>	计算目标元素的位置(仅计算)
phi	<code><result> = phi [fast-math-flags] <ty> [<val0>, <label0>], ...</code>	
zext..to	<code><result> = zext <ty> <value> to <ty2></code>	类型转换, 将<ty>的<value>的type转换为<ty2>

terminator insts

llvm ir	usage	intro
br	<code>br i1 <cond>, label <iftrue>, label <iffalse> br label <dest></code>	改变控制流
ret	<code>ret <type> <value>, ret void</code>	退出当前函数, 并返回值(可选)

(2) [github repo教程][<https://github.com/Evian-Zhang/llvm-ir-tutorial/blob/master/LLVM>]

1. 数据表示

- 栈上指针 比如在函数内部malloc

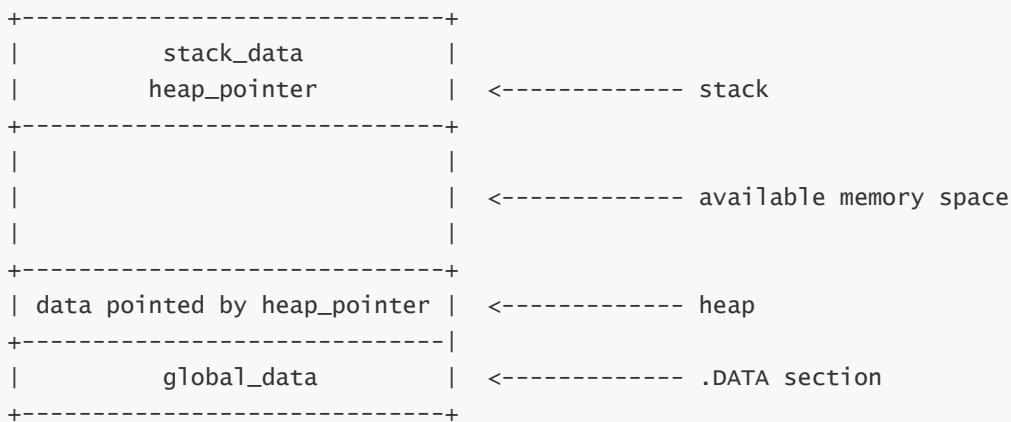
在栈上创建数组时，不能使用含有变量的表达式，如：int a[x+2];错误

原因：在栈上创建数组时编译器编译的时候就需要在栈上分配内存，可是有了变量以后，编译器就无法知道该分配多大的内存空间，故编译器会报错。但是定义一般变量如：int a；编译器会自动识别int占多大内存分配给他。

对比：如果是动态创建数组时（也就是在堆上创建数组时）可以出现变量如：new a [x+1]；正确；

原因：在堆上创建数组时，编译器不会在编译的时候为它分配内存，而是在程序运行的时候为它分配内存，我们可以知道，程序运行时变量的值就会明确是多少，故动态创建数组时可以出现变量；

堆里的是动态数组，是程序运行过程中动态加载的，而栈不一样，申请数组必须要是确定大小的数字，在编译时就要确定下来，如果你const int x=7就不会报错



堆中的数据不会独立存在 一般要不是栈上的数据要不是全局变量的数据

程序中的数据类型如下

- 寄存器中的数据
- 栈上的数据
- 数据区里的数据

数据区数据

```
i32 类型的只读全局变量和全局变量
@global_constant = constant i32 0
@global_variable = global i32 0
```

如果不考虑内存地址操作 栈可以认为是扩大的寄存器

*x86的一些特性

将两者中比较大的那个局部变量存储在栈上的 -4(%rbp) 上（由于x86_64架构不允许直接将内存中的一个值拷贝到另一个内存区域中，所以得先把内存区域中的值拷贝到eax寄存器里，再从eax寄存器里拷贝到目标内存中）

正是因为内存操作过于复杂 llvm引入了虚拟寄存器的概念

- 寄存器的使用

LLVM IR内部自动帮我们做了这些事。如果我们把所有没有被保留的寄存器都用光了，那么LLVM IR会帮我们把这些被保留的寄存器放在栈上，然后继续使用这些被保留寄存器。当函数退出时，会帮我们自动从栈上获取到相应的值放回寄存器内。

- 全局变量和栈上的变量都是指针

```
@global_variable = global i32 0
```

这个操作包含了alloca 但@global_variable存储 相当于一个寄存器

```
%1 = load i32, i32* @global_variable
store i32 1, i32* @global_variable
```

都用指针操作

nsw 是 "No Signed Wrap" 的缩写，表示无符号值运算

add i32指的是三个的类型

2.类型系统

聚合类型

数组

```
int a[4]
```

```
%a = alloca [4 x i32]
```

结构体

```
struct MyStruct {
    int x;
    char y;
};
```

```
%MyStruct = type {
    i32,
    i8
}
```

无论是数组还是结构体，其作为全局变量或栈上变量，依然是指针

对聚合类型进行操作：使用getelementptr

llvm是强类型语言

```
struct MyStruct {
    int x;
    int y[5];
};

struct MyStruct my_structs[4];
```

那么如果我们想获得 my_structs[2].y[3] 的地址，只需要

```
%MyStruct = type {
    i32,
    [5 x i32]
}
%my_structs = alloca [4 x %MyStruct]

%1 = getelementptr [4 x %MyStruct], [4 x %MyStruct]* %my_structs, i64 2, i32 1,
i64 3
```

我自己用clang编译的结果

```
@my_structs = dso_local global [4 x %struct.MyStruct] zeroinitializer, align 16
@my_structs_ptr = dso_local global %struct.MyStruct* getelementptr inbounds ([4 x
%struct.MyStruct], [4 x %struct.MyStruct]* @my_structs, i32 0, i32 0), align 8
@y = dso_local global i32 0, align 4
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void
()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_main.cpp, i8* null }]

; Function Attrs: noinline uwtable
define internal void @_cxx_global_var_init() #0 section ".text.startup" {
    %1 = load %struct.MyStruct*, %struct.MyStruct** @my_structs_ptr, align 8
    %2 = getelementptr inbounds %struct.MyStruct, %struct.MyStruct* %1, i64 2
    %3 = getelementptr inbounds %struct.MyStruct, %struct.MyStruct* %2, i32 0, i32
2
    %4 = load i32, i32* %3, align 4
    store i32 %4, i32* @y, align 4
    ret void
}
```

inbound是防止越界而使用的

`getelementptr` expects the type that you are indexing (without the pointer) as its first argument.

- 关于dso_local

dso_local

The compiler may assume that a function or variable marked as `dso_local` will resolve to a symbol within the same linkage unit. Direct access will be generated even if the definition is not within this compilation unit.

`load` 类型 源类型 地址 ,align 8

我们的指针只能够一层一层展开 暂时不能一层描述所有的情况

- 控制函数

比较指令

```
%comparison_result = icmp uge i32 %a, %b
bool comparison_result = ((unsigned int)a >= (unsigned int)b);
```

条件跳转

```
br i1 %comparison_result, label %A, label %B
```

无条件跳转

```
br label %start
```

basic block

- 一个函数由许多基本块(Basic block)组成
- 每个基本块包含：
 - 开头的标签 (可省略)
 - 一系列指令
 - 结尾是终结指令
- 一个基本块没有标签时，会自动赋给它一个标签

phi指令：

```
%y = phi i32 [1, %btrue], [2, %bfalse]
```

根据从哪一个block来进行赋值

前一个basic block是**%btrue**，那么返回1，如果前一个basic block是**%bfalse**，那么返回2

3. 函数

(四) 龙书

一. 引论

eg:

```
position=initial+rate*60
```

具体流程：词法分析：转化为标识符

语法分析：语法树 (syntax tree)

语法分析器：类型转换 类型检查

中间代码生成：

```
t1=inttofloat(60)
t2=id3*t1
t3=id2+t2
id1=t3
```

代码优化：

机器无关 机器相关优化

代码生成:

机器码

(MULF ADDF 等)

```
LDF R2 id3  
MULF R2,R2,#60.0  
LDF R1 id2  
ADDF R1, R1 R2  
STF id1, r1
```

#

(五)其他

一.助教名单



群聊成员



默认排序

群主、管理员(11人)



LV24 群主 许烨辰 arch TA



A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V



LV3 管理员 储浩天 compiler TA



LV41 管理员 蔡子逸 logic TA



LV5 管理员 季利恒 algebra TA



LV4 管理员 林春茹 arch TA



LV26 管理员 孙晨曦 arch TA



LV3 管理员 许楚豪 compiler TA



LV7 管理员 徐若凡 compiler TA



LV19 管理员 杨新宇 compiler TA



LV1 管理员 郑皓天 arch TA

W
X
Y
Z
#



LV12 管理员 张洪鑫 logic TA

C(2人)

cht xch

lcr scx zht

二.LLVM

LLVM是一套编译器基础设施项目，为自由软件，以C++写成，包含一系列模块化的编译器组件和工具链，用来开发编译器前端和后端。

三.研讨

fn root node 有什么用?

root node 什么作用

Java

AST
AST NMD????

visit accept

访问模式

root node

astBuilder.visit program

cd -> root.strDef.add('visited'))

list

) new node

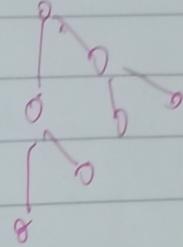
accept

fn root 代码翻译到字典表

symbol collector.

root node

visit



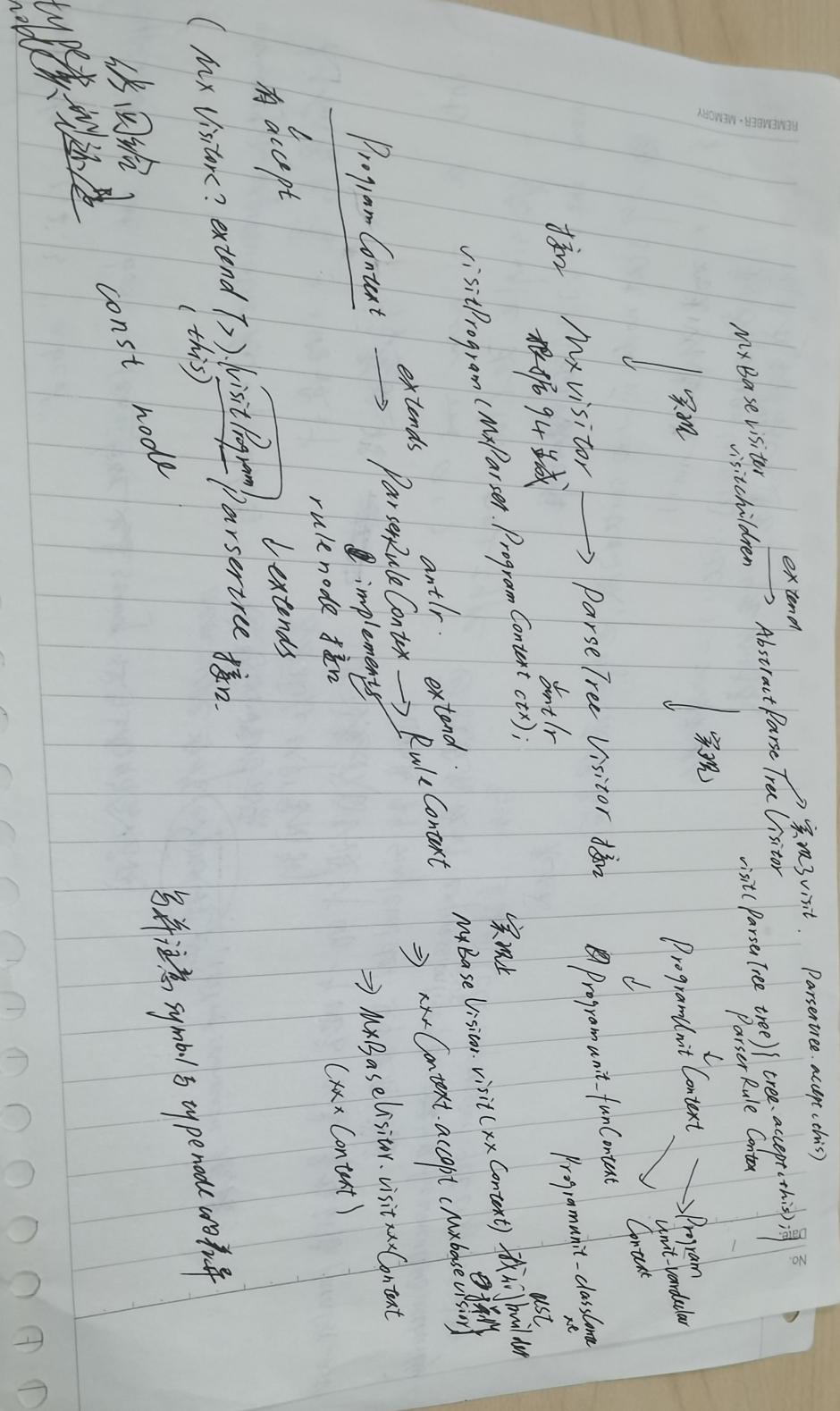
env

symbol

key
val

key
val

key
val



first swing 往上飛的發球

nen type $\frac{1}{2}$ (expressiv) sind scenario ~~die~~!

the ~~other~~ ^{other} ~~way~~ ^{way} to do
this ~~is~~ ^{is} to ~~exp~~ ^{exp} ~~and~~ ^{and} to do it ~~in~~ ⁱⁿ this ^{way} ~~to~~ ^{to} the parallel ~~last~~

~~单音节词~~ 中没有威德，
用的是多个 singlevalued. 其中没有威德
... onistic 节。(如 *reflexional* (as a building
block) 等)

② 我沒有設計
直接用 ctx.expression

形成，也可以有很多可

3-4 室工作

Date
No.

$$dB = \frac{\mu_0}{4\pi} \frac{i ds \sin\theta}{r^2} = \frac{\mu_0}{4\pi} \frac{i dB \sin\theta}{r^2}$$

安培定理
 $\int dB \cdot dS = \mu_0 I_{\text{enc}}$

$$\text{电场 } \oint E \cdot dS = \frac{q}{\epsilon_0} \quad \nabla \cdot \vec{E} = \frac{\rho}{\epsilon_0}$$

$$\oint B \cdot dS = 0 \quad \nabla \cdot \vec{B} = 0$$

无源场、引入电势

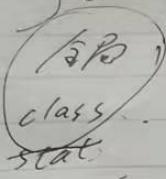
$$\oint B \cdot dL = \mu_0 I$$

有旋场

$$\nabla \times \vec{E} = 0$$

$$\nabla \times \vec{B} = \mu_0 I \rightarrow \text{电流密} \quad \text{度面积分}$$

全局 (stat) scope



slope

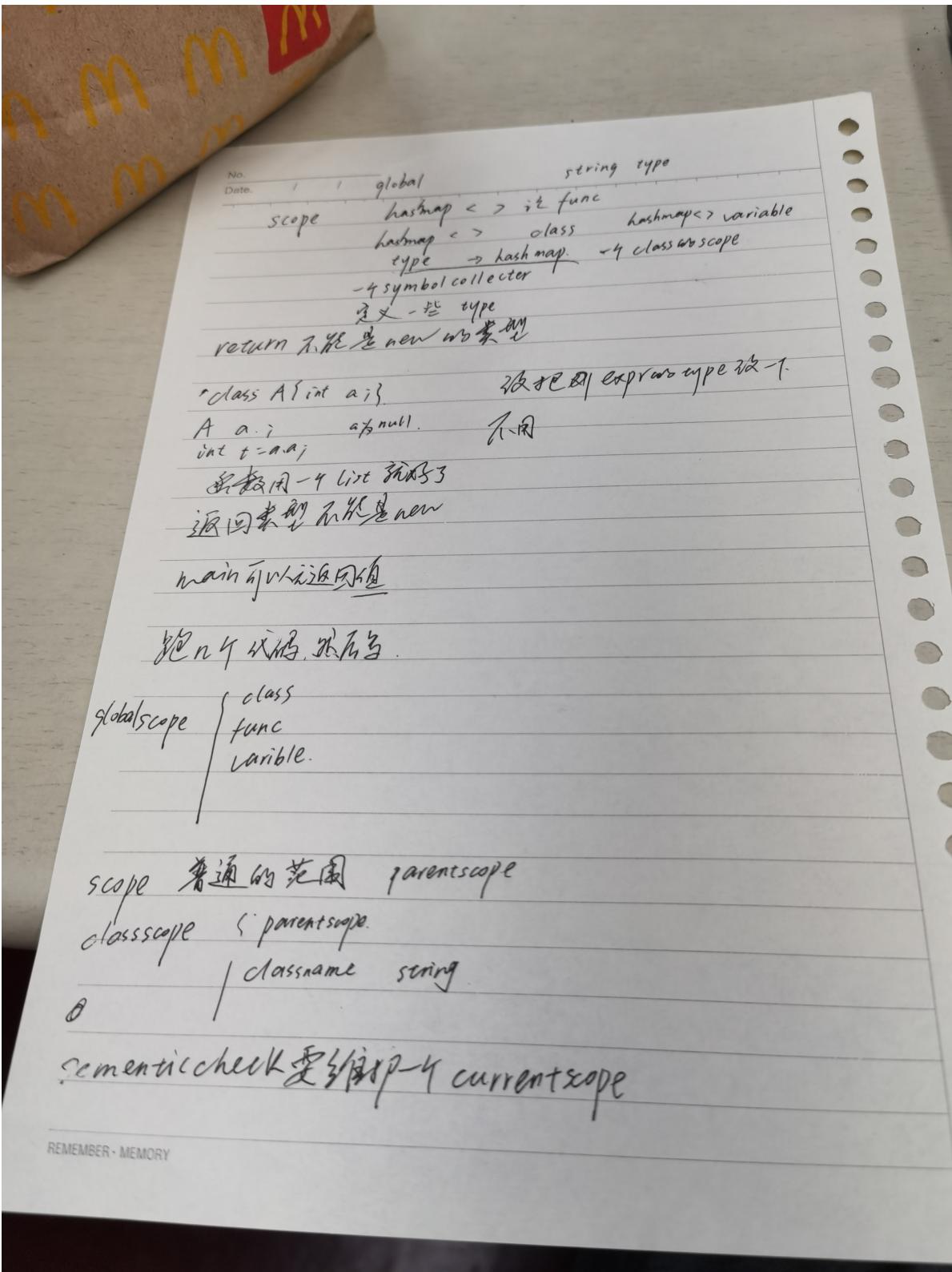
parent

null

slope
variable derive
↓
function dot
class dot

$b = c$

Wengu
文哥



四.Q

- 带有 phi 指令的 SSA IR
- 关于 ssa 还是 mem2reg...?
- 如何生成.s 遍历树
- 不用统计栈帧的大小 用类似于 pushq 的方法 codegen...?

`pushq` 指令：首先将栈指针 `%rsp` 减 8，然后将值写到新的栈顶地址，其等价于：

▼ MIPSASM

```
1 subq $8,%rsp    # Decrement stack pointer
2 movq %rbp,(%rsp) # Store %rbp on stack
```

- 用c的分配空间的方法
- 最后pointertype应该是多少byte....?

如果我的机子上能跑就是8byte 的

要是32位机子跑的话malloca的

```
step = new int[N][];
for (i = 0; i < N; i++) {
    step[i] = new int[N];
    for (j = 0; j < N; j++)
        step[i][j] = -1;
}

for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++)
        printlnInt(step[i][j]);
}
```

在这个点上会行为异常（关于pointertype的byttenum）

五.java

linkedhashmap 可以维护一个有序的hash表 比直接的hashmap多了有序性