CCCCC
C Complier Completed Crossing C

翁健

1er juin 2015

Table des matières

1 C 语言实现

首先说一下，相比于 Java 用 C 写的优势，功利的角度说，这么做有少量的 Bonus。其次 C 语言比 Java 要底层，写类似于编译器这种底层的东西比较“门当户对”（尹茂帆学长语）。正因为 C 的底层，代码可以写得紧凑、简单、粗暴。例如 Java 要写得洋洋洒洒的类型定义（继承、多态、封装），C 只要一个 struct 定义就能搞定，将父类中要用到的所有子类的域取一个并集，都定义在其中，然后多一个域来记录它具体是哪一种子类即可。这并不比 Java 麻烦多少，相反的 Java 还要用 instanceof 来判断属于哪一个子类，显得十分冗长。

下面我按照各个 Phase 来说我的实现。

2 语法分析

我手写的 Parser，使用的 Parser 算法是递归下降，精髓在于特判，通过仔细观察文法就能够知道进入下一个 Token 的时机，大多数时候是确定的，但是中国有一句古话叫做“万事开头难”，其实最麻烦的地方在于开头，判断这是一个全局变量申明还是一个函数定义，然后如果是全局变量申明时候是否初始化赋值。还有一个不确定的地方是 cast expression，因为遇到括号不知道里面是一个 expreesion 还是一个类型名。

利用递归下降的另一个好处在于可以随心所欲地掌控自己想要在其中插入的代码，如果用类似于 bison 一类的软件进行语法分析的话，里面能够插入的语句相对比较简单（只能建一个 CST 或者 AST），但是如果用手写的 parser 可以在其中插入自己想要的代码，实现一个语法制导的编译器，这样的编译器结构紧凑代码易读，同时也能够实现与符号表，控制流图的交互，一趟获取编译程序所需的大部分知识，我可以在一遍 parser 完了以后得到控制流图，和每一条控制流图里面的中间代码（为此我省略了 AST，用在递归下降的各个过程之间行走代替 CST）。

2.1 Pretty Printer

按照 Engineering a compiler 上的说法，一个 pretty printer 应该基于 parser，我手写的 parser 就有相当的优势，因为 lexer 不能很好地理解代码的意图，而 parser 能够轻易的做到这一点。

3 语义分析

每次在递归下降的过程中，如果 parse compound statement 或者 statement 就传入是否在循环中以判断 break 和 continue（注意一个细节，continue 是跳转到循环的条件判断处，而 for 语句是跳转到上面的第三段，然后第三段再跳转到条件判断处。

然后就是检查类型了，检查 . 是否不是指针且存在这个成员，检查 -> 是否是一个一重指针。检查各个二元运算是否合法，是否是左值。

typedef，如果使用单纯的 parser 不进行语义分析，当依法分析器遇到一个 identifier 的时候将无法分辨它是一个变量还是被定义的类型名，但是如果使用手写 parser 语法制导的编译器实现方式，可以在语法制导的过程中进行与符号表的交互，把 \typedef 后面的所有 declarators 都当做变量塞入符号表。不定长数组，只要在 parser_initializer 的时候，把第一维度不定长算出来就可以了。我为了使得代码显得简洁优美，所有的 struct 都用了 typedef 来定义。

4 中间表示设计

4.1 控制流图

正如前面所提到的，我利用控制流图来进行中间表示，每一个基本块都以一个 label 开头，一个跳转语句结尾。这样的中间表示有很多优越性，比如可以建 SSA（尝试未果），可以先简单粗暴地建出 IR 然后方便地进行控制流提升和控制流合并。

4.2 三地址码

我的三地址码很普通，分为移动操作、二元运算、控制流语句、系统调用和跳转语句。

4.3 变量类型

我产生的变量分为三类，程序中实际存在的变量，计算产生的临时变量和数组引用变量。其中实际存在的变量由 parse primary 产生，计算产生的临时变量由二元、一元运算产生，而数组引用是为了解决这样的一个矛盾而存在的：一个变量如果存在于赋值语句左边我们需要的是它的地址，而存在于赋值语句的右边我们要的是它的值，而如果一个变量它的类型是数组引用，那么我们可以发现，它马上将要被用到值还是地址是不确定的，这时候就把它设为数组引用类。当它出现在等号右边的时候就把它的值读出来，当它出现在左边的时候，就把它的地址读出来。

4.4 条件跳转

因为在 parse 表达式的时候不知道它是一个条件表达式还是一个求取值的表达式，所以我都把它们当做求取值的表达式来做了。然后结束的时候用等于零不等于零来判断一下。这样子会多出来几条冗余的语句，在后面的窥孔优化中可以优化掉。

5 代码生成

有了中间表示之后其实变成最终的代码已经很方便了，只要分析好中间表示中各个地址的类型，然后进行相应的 load/stroe 操作就可以了。我没有写寄存器分配，主要原因是我对窥孔优化还抱有幻想。我有专门的过程，叫 IR2XXX 用来进行代码生成。

6 优化

6.1 未优化

Bulgarian_solitaire-5110379024-wuhang	3806888
basicopt1-5100309127-hetianxing	3176345
board-5100379110-daibo	49579
expr-5110309085-jintianxing	110410
factor-5090379042-jiaxiao	613
gcd-5090379042-jiaxiao	1218
hanoi-5100379110-daibo	1052934
hanoi2-5100309153-yanghuan	2682283
hashmap-5100309127-hetianxing	1255761
heapsort-5100379110-daibo	45544244
hello-5090379042-jiaxiao	20
horse-5100309153-yanghuan	20237471
horse2-5100309153-yanghuan	23226681
horse3-5100309153-yanghuan	27210467
hpplus-5100309153-yanghuan	13924
inlinecheck-5100309153-yanghuan	870
magic-5100309153-yanghuan	3780102
manyarguments-5100379110-daibo	2666
manymanyarguments-5100379110-daibo	388
maxflow-5100379110-daibo	38482421
multiarray-5100309153-yanghuan	17970
nqueencon-5100379110-daibo	20142
pi-5090379042-jiaxiao	35697211
plus-5100309153-yanghuan	526
prime-5100309153-yanghuan	10445485
prime2-5100309153-yanghuan	9274693
qsort-5100379110-daibo	10548080
queenbitwise-5110379024-wuhang	285247
queens-5100379110-daibo	2433988
self-5090379042-jiaxiao	4410
selfref-5090379042-jiaxiao	253
spill-5100379110-daibo	3111
spill2-5100379110-daibo	121642
struct5-5110379024-wuhang	185536
struct6-5100309127-hetianxing	916
superloop-5090379042-jiaxiao	7291127
tak-5090379042-jiaxiao	2531750
treap-5110309085-jintianxing	15626134
twinprime-5090379042-jiaxiao	3378701
varptr-5100309127-hetianxing	14752

6.2 合并跳转指令

Bulgarian_solitaire-5110379024-wuhang	3452079
basicopt1-5100309127-hetianxing	2863194
board-5100379110-daibo	45342
expr-5110309085-jintianxing	110217
factor-5090379042-jiaxiao	532
gcd-5090379042-jiaxiao	1090
hanoi-5100379110-daibo	928621
hanoi2-5100309153-yanghuan	2366964
hashmap-5100309127-hetianxing	1097449
heapsort-5100379110-daibo	43223915
hello-5090379042-jiaxiao	20
horse-5100309153-yanghuan	17030803
horse2-5100309153-yanghuan	19873283
horse3-5100309153-yanghuan	23495415
hpplus-5100309153-yanghuan	12765
inlinecheck-5100309153-yanghuan	810
magic-5100309153-yanghuan	3307651
manyarguments-5100379110-daibo	2438
manymanyarguments-5100379110-daibo	369
maxflow-5100379110-daibo	34381090
multiarray-5100309153-yanghuan	16722
nqueencon-5100379110-daibo	16963
pi-5090379042-jiaxiao	35687535
plus-5100309153-yanghuan	490
prime-5100309153-yanghuan	9671064
prime2-5100309153-yanghuan	8620056
qsort-5100379110-daibo	9564551
queenbitwise-5110379024-wuhang	270846
queens-5100379110-daibo	2117101
self-5090379042-jiaxiao	3874
selfref-5090379042-jiaxiao	234
spill-5100379110-daibo	2997
spill2-5100379110-daibo	111914
struct5-5110379024-wuhang	173814
struct6-5100309127-hetianxing	859
superloop-5090379042-jiaxiao	4347946
tak-5090379042-jiaxiao	2289199
treap-5110309085-jintianxing	13862343
twinprime-5090379042-jiaxiao	3025616
varptr-5100309127-hetianxing	13657

6.3 合并不必要的基本块

Bulgarian_solitaire-5110379024-wuhang	3328225
basicopt1-5100309127-hetianxing	2798508
board-5100379110-daibo	44192
expr-5110309085-jintianxing	110151
factor-5090379042-jiaxiao	526
gcd-5090379042-jiaxiao	1069
hanoi-5100379110-daibo	895879
hanoi2-5100309153-yanghuan	2285063
hashmap-5100309127-hetianxing	1070446
heapsort-5100379110-daibo	42608039
hello-5090379042-jiaxiao	20
horse-5100309153-yanghuan	16566199
horse2-5100309153-yanghuan	19357867
horse3-5100309153-yanghuan	22832282
hpplus-5100309153-yanghuan	12422
inlinecheck-5100309153-yanghuan	790
magic-5100309153-yanghuan	3212704
manyarguments-5100379110-daibo	2366
manymanyarguments-5100379110-daibo	363
maxflow-5100379110-daibo	33131579
multiarray-5100309153-yanghuan	16256
nqueencon-5100379110-daibo	15928
pi-5090379042-jiaxiao	35116133
plus-5100309153-yanghuan	478
prime-5100309153-yanghuan	9486512
prime2-5100309153-yanghuan	8470280
qsort-5100379110-daibo	9305480
queenbitwise-5110379024-wuhang	264740
queens-5100379110-daibo	2041843
self-5090379042-jiaxiao	3728
selfref-5090379042-jiaxiao	228
spill-5100379110-daibo	2961
spill2-5100379110-daibo	108842
struct5-5110379024-wuhang	170373
struct6-5100309127-hetianxing	842
superloop-5090379042-jiaxiao	4178541
tak-5090379042-jiaxiao	2289193
treap-5110309085-jintianxing	13734717
twinprime-5090379042-jiaxiao	2921421
varptr-5100309127-hetianxing	13282

6.4 将 printf 中常量特判

Bulgarian_solitaire-5110379024-wuhang	2408253
basicopt1-5100309127-hetianxing	2798317
board-5100379110-daibo	32248
expr-5110309085-jintianxing	110151
factor-5090379042-jiaxiao	343
gcd-5090379042-jiaxiao	1069
hanoi-5100379110-daibo	895696
hanoi2-5100309153-yanghuan	2285063
hashmap-5100309127-hetianxing	1070446
heapsort-5100379110-daibo	40770431
hello-5090379042-jiaxiao	20
horse-5100309153-yanghuan	16566015
horse2-5100309153-yanghuan	19357683
horse3-5100309153-yanghuan	22832098
hpplus-5100309153-yanghuan	6212
inlinecheck-5100309153-yanghuan	100
magic-5100309153-yanghuan	3186569
manyarguments-5100379110-daibo	160
manymanyarguments-5100379110-daibo	180
maxflow-5100379110-daibo	33131579
multiarray-5100309153-yanghuan	10184
nqueencon-5100379110-daibo	15088
pi-5090379042-jiaxiao	32861733
plus-5100309153-yanghuan	64
prime-5100309153-yanghuan	8358224
prime2-5100309153-yanghuan	8470280
qsort-5100379110-daibo	7465480
queenbitwise-5110379024-wuhang	264557
queens-5100379110-daibo	2041843
self-5090379042-jiaxiao	3728
selfref-5090379042-jiaxiao	45
spill-5100379110-daibo	1857
spill2-5100379110-daibo	14634
struct5-5110379024-wuhang	161637
struct6-5100309127-hetianxing	659
superloop-5090379042-jiaxiao	4178358
tak-5090379042-jiaxiao	2289010
treap-5110309085-jintianxing	13368534
twinprime-5090379042-jiaxiao	2921421
varptr-5100309127-hetianxing	13282

7 Bootstrap

我的编译器支持 bootstrap，存在一个版本（位于文件夹 bootstrap 中），用 clang 编译出来的可执行文件以自身作为输入，得到的 mips 再度源码作为输入可以得到自身。为了能够支持 bootstrap 我自己手工实现了类似于 sprintf 功能的 str_cat（无副作用地连接两个字符串）、my_assert（类似于 c 的 assert，根据条件抛出异常）、my_strcmp（与 C 的 strcmp 相同）

8 额外的工作

支持函数、类型的预定义，支持在 struct 里面套有匿名的 struct 或者 union，可以访问。都在 bootstrap 中得以体现。

9 各个类的定义

```
struct Token {
    int type; //Token的类型，Id、字符常量、数字常量、运算符、字符串常量
    int col, row; //Token所在的行列
    char *_literal; //Token的字面值，如果是字符串包含""
    union { //如有，Token所代表的值
        int int_val; //其实下面两者是一样的，这里写的有点冗长了
        int char_val;
        char *str_val; //字符串转义之后的每一位的值
    };
};
struct Block {//保证每个基本快以一个label开头，0-1条件跳转与一个无条件跳转结尾
    int id, in_deg; //基本块的编号；基本块的入度，优化用
    int ins_size, buffer_size; //基本块的IR指令vector
    Instruction *ins;
    Block *non_condi; //无条件或者跳转条件失败的跳转label
    Block *condi; //条件跳转指令达成跳转的label
};
```

```
struct Function {
    Type *type; //函数的返回值类型
    int level; //指针的重数
    int tmp_cnt; //该函数中用到的临时变量计数，IR时用
    char *id; //函数的名字
    Identifier *args; //参数表，参数表里面的变量分存两份，还有一份在函数所对应的环境中
    Function *nxt; //自带链表
    Block *block; //函数的开头的基本块
    Block *end; //函数的结束基本块
    ReturnType *regs; //函数中用到的临时变量的链表
};
struct Instruction { //三地址码
    int ins; //指令类型
    int ord; //总的第ord条指令
    ReturnType *des, *a, *b; //三地址
};
struct ReturnType { //由表达式运算产生
    int ret_type; //分为三类，伪寄存器、数组的尽头和常量
    int is_left; //是否为左值
    int const_val; //如果为常量，值
    Function *func; //如果是表达式中的一个函数，对应函数的引用
    ReturnType *nxt; //在所对应的Function中临时变量的列表
    Identifier *ref; //运算临时变量记录其类型/否则记录其所对应的实际变量
    int reg_num; //临时变量的编号
    int sp_offset; //在栈中的偏移量
    Function *belong; //属于哪一个函数
    Register *reg; //寄存器分配没写完T_T
};
struct Declarator { //与语法中Declrator的定义同
    char *_literal; //字面值
    int level; //*个数
    Declarator *nxt; //自带链表，函数的参数表中用到
    int is_func; //有没有后面的()，如果有是一个函数
    Identifier *args; //函数的参数表带有类型，所以要用id类记录
    Array *dim; //数组的维度
};
struct InitPair { //初始化列表的类
    int pos, num; //所在下标，初始化成的值
    char *label; //或者字符串指针
    InitPair *nxt; //自带链表
};
```