编译原理实验

张家荣　张翔宇　钟睿智　张瑞康

中国科学技术大学
University of Science and Technology of China

主要内容

- print
- 数组
- 取地址符与指针
- 指针数组
- 作用域算符::

01

print

print

- print(expression1, expression2, ...)
- 结束后会打印换行符

print

- print作为保留字
- 为了增强可读性，用单独的PRT指令实现
- PRT 0 0： 打印并弹出栈顶元素
- PRT 0 1： 打印换行符\n
- 每条表达式求值后，添加PRT 0 0
- 遇到右括号后，添加PRT 0 1

02

数组

数组的实现

- 新增了标识符 SYM_ARRAY，用来识别数组变量。
- 数组按照连续空间存储（以行为主）
- 每次识别数组变量时有两种情况：
- 1、该数组的索引完全，即‘[]’个数和其维数相等，此时返回其值
- 2、索引不完全，此时返回计算的地址

数组的实现

- 每次getsym()得到数组变量之后会得到每一维对应的值（定义时则为该维度数量，访问时则为下标）

数组的实现

```
case ID_ARRAY:
    mk = (mask*) &table[i];
    int tmp=0;
    int tmp_address=0;
    int not_full_array = 0;
    while(mk->len[tmp] && tmp<num_dim_tmp){
        if(len_left[tmp] >= mk->len[tmp])
            error(29);
        if(!tmp)
            tmp_address = len_left[tmp];
        else
            tmp_address = tmp_address*(mk->len[tmp-1]) + len_left[tmp];
        if(!len[tmp])
            not_full_array = 1;
        tmp++;
    }
    if(!is_addressof){
            if(not_full_array)
                gen(LIT, 0, mk->address + tmp_address);
            else
            gen(LOD, level - mk->level, mk->address + tmp_address);
        }else
            gen(LIT, 0, mk->address + tmp_address);
    break;
```

03

取地址符与指针

取地址符&的实现

- 在 factor() 处判断，若有取地址符号，标记一个 flag，计算栈顶的值时便采用 LIT, 0, mk->address
- 例如：（实际上每一个变量/数组/指针处都要判断）

```
case ID_VARIABLE:
    mk = (mask*) &table[i];
    if(!is_addressof)
        gen(LOD, level - mk->level, mk->address);
    else
        gen(LIT, 0, mk->address);//address of
    break;
```

指针的实现

- 定义了指针类型 SYM_POINT，在变量定义的时候若检测到*则下一个 enter() 的变量是指针类型。
- 对于若干个*，无论是定义处还是 factor() 处都会先统计 * 的个数，记录在某个全局变量上

指针的实现

- 并且新增了两种中间语言：LODI 和 STOI，均为间接访存语言。
- 其中：
- LODI： stack[top] := stack[stack[top-1]]
- STOI： stack[stack[top]] := stack[top-1]

指针的实现

- 每次当需要某个指针的时候，例如**r，先统计其*个数，这里为2重指针。
- 先LOD把r的值传到栈顶，再LODI一次把r的值对应的地址 的值传到栈顶。之后对栈顶操作即可。

指针的实现

```
case ID_POINTER:
    mk = (mask*) &table[i];
    if(sym == SYM_IDENTIFIER){
        if(!is_addressof){
            gen(LOD, level - mk->level, mk->address);
            while(pointer_length){
                gen(LODI, 0, 0);
                pointer_length--;
            }
        }else{
            gen(LIT, 0, mk->address);
        }
```

04

指针数组

指针数组

- 指针数组放在指针类型下，即标识为SYM_POINT
- 当识别到*，代表其为指针，getsym()后发现后面是一个数组，则按照指针数组的方式处理。
- 实际上是把指针的处理方式和数组的方式相结合，特殊判断

指针数组

```
case ID_POINTER:
    mk = (mask*) &table[i];
    if(sym == SYM_IDENTIFIER){
        if(!is_addressof){
            gen(LOD, level - mk->level, mk->address);
            while(pointer_length){
                gen(LODI, 0, 0);
                pointer_length--;
            }
        }else{
            gen(LIT, 0, mk->address);
        }
    }else
    if(sym == SYM_ARRAY){
        int tmp=0;
        int tmp_address=0;
        while(mk->len[tmp] && tmp<num_dim){
            if(len_left[tmp] >= mk->len[tmp])
                error(29);
            if(!tmp)
                tmp_address = len_left[tmp];
            else
                tmp_address = tmp_address*(mk->len[tmp-1]) + len_left[tmp];
            tmp++;
        }
        if(!is_addressof){
            gen(LOD, level - mk->level, mk->address + tmp_address);
            while(pointer_length){
                gen(LODI, 0, 0);
                pointer_length--;
            }
        }else{
            gen(LIT, 0, mk->address + tmp_address);
        }
```

一些小细节

- 原程序必须按照 Const.....;Var.....;的顺序进行定义，甚至不能有两行 Var。将block中的两个 if 语句改为 while 语句即可实现多行 Const 和多行 Var。
- 但是依旧需要按照Const到Var的顺序来定义。将外层改为while应该就可以忽略顺序了。

一些小细节

- 原程序的 set 键值是从小到大索引的。往 first 集合添加新标识符时要注意插入顺序，不能在 SYM_NULL 后面
- LODI 和 STOI 需要考虑base(stack, b, i.l)。
- 利用全局变量访问信息时，在 factor() 中要用临时变量进行保留，否则递归过程中值会改变

没有做到的

- *(*(*(brr + 1))) 类型的连续访存。
- 由于设计时数组只记录了头位置，而且是连续存放。所以只能支持一维索引。这是设计数组时的缺陷。
- 推测数组多记录各维度的指针可能可以实现。

05

作用域算符::

作用域算符 ::

- 使用语法：形如::i，::p1::i，::p1::p2::i等用来访问在过程嵌套声明中的外层过程所声明的同名变量。
- 其中::i表示最外层主程序main中声明的i.
- ::p1::i表示最外层声明的过程p1中声明的i.
- 通式：::{ident::}*ident，闭包内的ident为过程，外为变量。

具体实现

- 修改first(factor)集合
- 将作用域算符::作为单目运算符，修改getsym函数。
- 添加position_field函数用来寻找目标过程层次下同名变量在表中的序号
- 修改factor函数，记录所求过程并利用postion_field函数得到同名变量层次与相对地址，生成LOD指令

演示

example2.txt

```
1 var i;
2 procedure p1;
3   var i;
4   procedure p2;
5     var i;
6     procedure p3;
7       var i;
8       begin
9         print(::i+3);
10        i:=::p1::i+3;
11        i:=::p1::p2::i+3
12      end;
13    begin
14      i:=3;
15      call p3;
16    end;
17  begin
18    i:=2;
19    call p2;
20  end;
21 begin
22  i:=1;
23  call p1;
24 end.
```

```
Begin executing PL/0 program.
1
2
3
prt:4
5
6
End executing PL/0 program.
```

谢谢！

中国科学技术大学
University of Science and Technology of China