Homework 7

PL/0 Exercise1

PB17000297 罗晏宸

October 29 2019

阅读 PL/0 编译器相关文档，完成以下任务：

在 PL/0 编译器中，函数`interpret()`在解释指令 LOD/STO 时的语义代码如下：

```
case LOD: // 指令格式 (LOD, l, a)
    stack[++top] = stack[base(stack, b, i.l) +
       i.a];
    break;
case STO: // 指令格式 (STO, l, a)
    stack[base(stack, b, i.l) + i.a] = stack[top];
    printf("%d\n", stack[top]);
    top--;
    break;
```

(a) 你“扩展”PL/0 编译器，添加了 LEA/LODA/STOA 等指令。格式为：(LEA, l, a)，(LODA, 0, 0) 和 (STOA, 0, 0)。其中“取地址”指令 LEA 用来获取名字变量在“运行时栈-stack”上“地址偏移”。而“间接读”指令 LODA 则表示以当前栈顶单元的内容为“地址偏移”来读取相应单元的值，并将该值存储到原先的栈顶单元中。而“间接写”指令 STOA 则将位于栈顶单元的内容，存入到次栈顶单元内容所代表的栈单元里，然后弹出栈顶和次栈顶。给出这样的 LODA/STOA/LEA 指令的语义代码。

解 语义代码如下

```
case LODA: // 指令格式 (LODA, 0, 0)
    stack[++top] = stack[stack[base(stack, b, i.l)
       + i.a]];
    break;
case STOA: // 指令格式 (STOA, 0, 0)
    stack[stack[top - 1]] = stack[top];
    printf("%d\n", stack[top]);
    top = top - 2;
    break;
case LEA: // 指令格式 (LEA, l, a)
    stack[++top] = base(stack, b, i,l) + i.a;
    break;
```

(b) 现在继续扩展 PL/0 编译器。假设你实现了若干 C 风格的表达式、类型及其声明体系，并可编译如下程序：

```
int main()
{
int  i;
int* q;
int* a[10];
int* (*b[10])[10];
int* (*(*p)[10])[10];

i = 100; q = &i; a[1] = q; b[1] = &a; p = &b;

cout <<____p[0]____<< endl; // 输出 100，待补全

cout <<_____*p_____<< endl; // 输出 100，待补全

} // 程序
```

- 给出变量 a 和 p 的类型表达式。
  注意，int 即为类型表达式。指针类型表示为 pointer(T1)，T1 为指针所指向对象的类型表达式，数组类型表示为 array(number,T2)，数组元素的个数 number 为常量值，T2 为数组元素的类型表达式。

- 根据 PL/0 编译环境设定，上述程序中分配的总变量空间是多少？各个变量在活动记录中“地址偏移”是多少？

- 两条输出语句中不同的表达式各自仅包含唯一的名字变量 p。根据你的 C 语言知识，补全这两处输出语句中的源代码。

- 给出一个上述下划线处源代码对应的 PL/0 代码（两处输出语句可任选其一产生 PL/0 代码）。如需使用算术运算，可直接给出，例如，加法(ADD, 0, 0)，乘法(MUL, 0, 0)，以及加载常数于栈顶 (LIT, 0, 100)等指令。

解

- 变量 a 是一个含 10 个元素的指针数组，类型表达式为

  array(10, pointer(int))

  变量 p 是一个指针，指向一个含 10 个元素的数组，数组的每个元素是一个指向含 10 个元素数组的指针，被指向数组的每个元素是一个指向整型变量的指针，类型表达式为

  pointer(array(10, pointer(array(10, (pointer(int))))))　。

- 为变量 i 分配 1 字节，变量 q 分配 1 字节，变量 a 分配 $1 \times 10 = 10$ 字节，变量 b 分配 $1 \times 10 = 10$ 字节，变量 p 分配 1 字节，故分配的总变量空间为 $1 + 1 + 10 + 10 + 1 = 23$ 字节；
  各个变量在活动记录中“地址偏移”分别是变量 i 偏移 0 字节，变量 q 偏移 $0 + 1 = 1$ 字节，变量 a 偏移 $1 + 1 = 2$ 字节，变量 b 偏移 $2 + 10 = 12$ 字节，变量 p 偏移 $12 + 10 = 22$ 字节。

- 代码补全如下

```
cout <<*(*p[0][1])[1]<< endl; // 输出 100
cout <<**(*(*p + 1) + 1)<< endl; // 输出 100
```

- 对应的 PL/0 代码如下

```
(LIT, 0, 100)
(STO, 0, 0)
(LEA, 0, 2)
(LIT, 0, 1)
(ADD, 0, 0)
(LOD, 0, 1)
(STOA, 0, 0)
(LEA, 0, 22)
(LEA, 0, 12)
(STOA, 0, 0)
(LEA, 0, 22)
(LODA, 0, 0)
(LIT, 0, 1)
(ADD, 0, 0)
(LODA, 0, 0)
(LODA, 0, 0)
(LIT, 0, 1)
(ADD, 0, 0)
(LODA, 0, 0)
```

(c) 再扩展 PL/0 编译器，你添加了“引用”声明及处理。一个引用变量也具有一个地址单元，其中存储着被“引用”的其他变量的地址偏移。
考虑如下程序片段：

```
int* &r = ⋯ // r 是一个引用变量，被引用对象是一个 int 指针变量。
int func(int *i, int* &j, int k); // 函数 func 声明
```

对于函数调用：`func(r, r, *r)` 分别给出计算三个实参的“值”到 stack 栈顶的 PL/0 代码。假设 `r` 的地址偏移为 3。

解 PL/0 代码如下

```
(LOD, 0, 3)
(LOD, 0, 3)
(LEA, 0, 3)
```