**Report for Lab03 – Garbage Collector**

**Stu:金泽文 No.PB15111604**

使用语言：C++

环境 linux64 g++ 6.2.0 –std=c++11

运行方法：

编译方法：

make 或者

g++ Prj.cpp Term.cpp -o p -std=c++11 –w

运行方法 ：

./p 直接运行

./p 加上任意一个参数，导出处理之前与处理之后的语法树结构，可发现命名变化。

./p 加上任意两个参数，导出处理之前与处理之后的语法树结构，可发现命名变化。输出行号。（调试用）

测试样例见文件examples.txt

下面是关于gc算法的说明。

**Garbage Collector的实现**

具体思路：

由于我的解释器整体是在语法树上进一步生成的，所以思路跟别人的“活动记录”实现方式有比较大的不同。

典型的gc算法，如标记-清除，引用计数，复制算法，等统统都没有用。由于我们要实现的语言功能很单一，没有指针的概念，所以只需要在解释的过程确定之后不会再用到即可清除，释放空间。

我是针对每一块语法结构而设计的算法。

①针对**函数调用**，②针对Assign，Read，Print，If，While等**Command**。

为了释放空间，我设计了两个函数，deleteMem和deleteTree，后者被前者调用，前者通用于以上所有的情况释放节点所用。

①针对**函数调用**：由于我的函数是在每个的apply和call的sons链表中push\_back一个子树来复制函数体并且调用，所以我释放函数是在函数调用完毕之后释放sons.back()。

②针对Command：由于这些command在一次解释之后就不在有作用，所以直接释放，通过deleteMem。

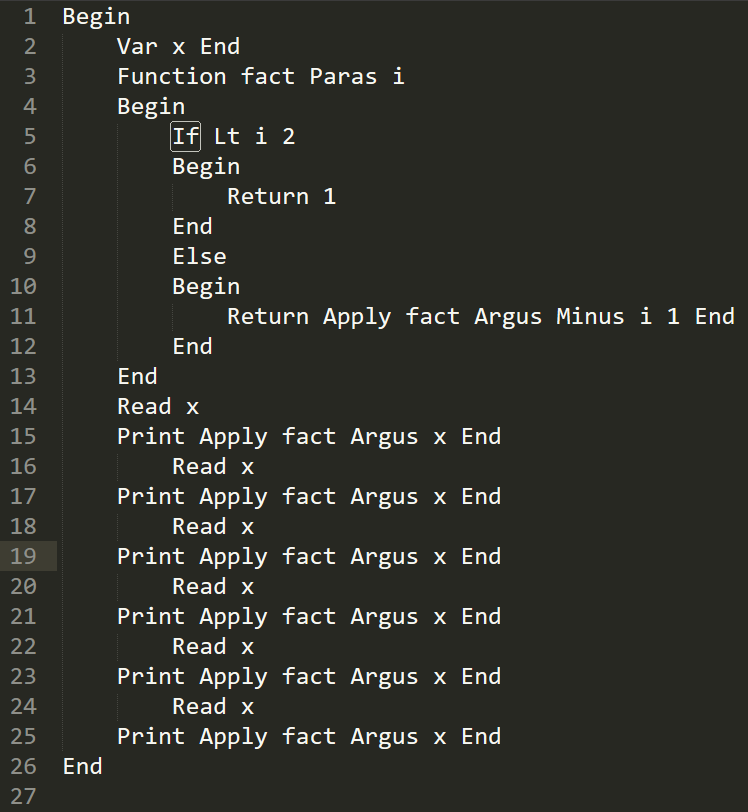
需要注意的是，由于While的存在，所以在释放的过程中，要考虑之后是否要复用，所以，call 之后利用underWhile函数判断是否要复用，如果不服用，则把call节点也释放掉；command解释之后，也利用underWhile函数判断是否需要复用，如果需要复用，则不能释放。

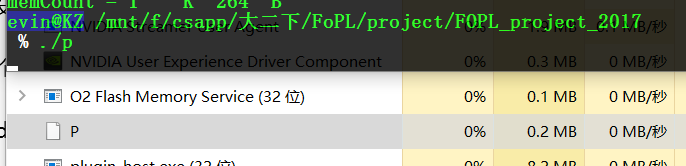
另外，对于Block中的变量作用域的问题，由于block只能在if，while，call，apply中出现，而这四个我已经考虑了，所以不需要进一步考虑。

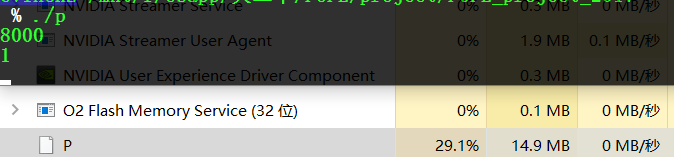
最后，我写了几个测试样例，并通过设置maxMem（内存峰值）,memCount（当前内存）,totalMem（总共分配过的内存）三个变量来表现我的gc的效果。

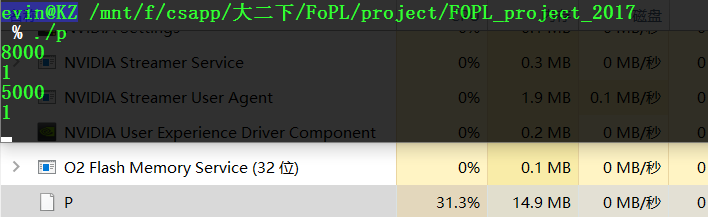
针对测试样例，我是在BashOnWindows上跑的，内存占用是在windows的任务管理器上观察的。

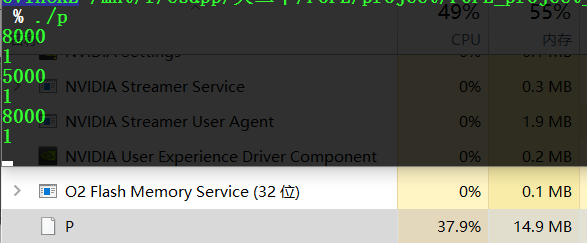
样例1：

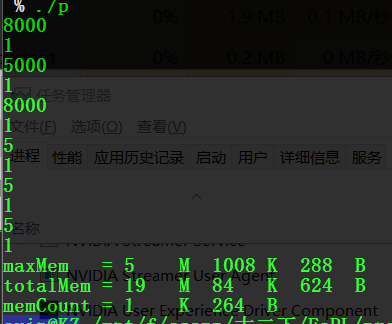


 开始：等待输入：目前内存0.2MB

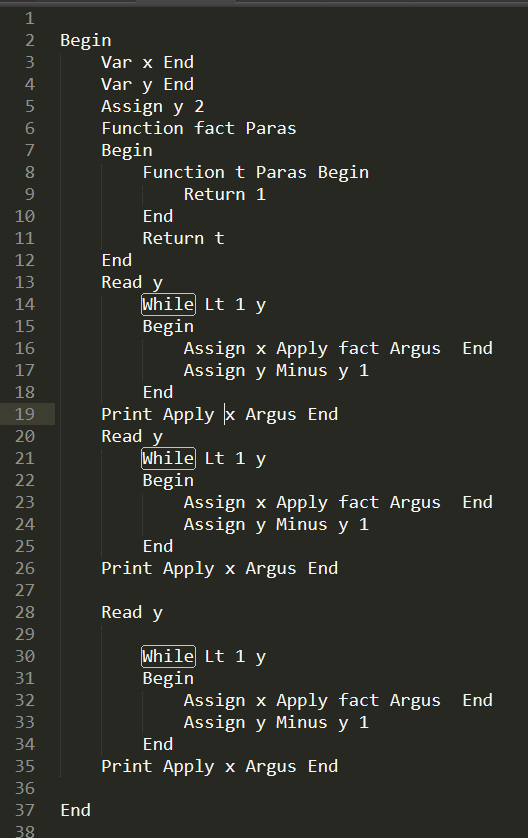
 输入8000，输出1，等待输入：可以看到，目前内存14.9MB

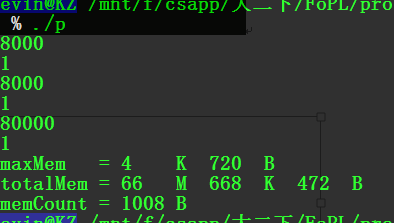
 再输入5000，输出1，等待输入：可以看到，内存还是14.9MB，没有增加。

 再输入8000，输出1，等待输入：同上，没有增加。

 最终：

样例2：



输入8000,8000,80000.

效果如右图。

下面是前两次实验过程的记录，供助教参考。

**前两次实验过程的部分文档记录**

对于prj1，我实现的方式与细节：

1.函数：

return 0：

考虑到默认的return 0，在语法树处理之后自动添加return 0，利用的是setFunction和addReturn函数。

return 的实现：

利用searchFun函数找到调用的地方，即call和apply的节点。并将值赋给这给节点的number。

declare：

无视declare

call，apply：

找到调用的函数，根据searchName

对于Argus，参数值通过expr得到，并赋值。

对于function block部分，采用deepcopy的模式。

并且由于要copy，所以参数列表的参数名必须与声明函数时一致。所以要换名字。

总体用callFunction函数实现

2.static scope:

将所有的变量名按照所声明的指针值重命名，

先解决使用变量时的名字在。

call apply 或者expr时

的变量名根据static\_searchName和changeUsedName函数

再解决def变量。

根据changeDefName函数。

遍历一律采用先序遍历。

3.语法树结构：

Begin End -> k=Block,s=Declaration

Var x End -> k=Command,s=Declaration

-> k=Name,s=Declaration,number,name

Assign x 1 -> k=Command,s=Assign

-> k=Name,s=Declaration,number,name

-> k=Expr,s=Number,number,name=""

Assign x y -> k=Command,s=Assign

-> k=Name,s=Declaration,number,name

-> k=Expr,s=VarName,number,name="y"

Function g Paras i-> k=Function,s=Declaration

-> k=Name,s=Declaration,number,name

-> k=Name,s=Declaration,number,name

-> k=Block,s=Declaration

->...

Read x -> k=Command,s=Read

-> k=Name,s=Declaration

Call h Argus y End -> k=Command,s=Call,

-> k=Name,s=Declaration,name

-> k=Expr,s=VarName,number,name

Prj2 高阶：

things to be solved

1.return function

--只有apply的时候才会用到return

应该在apply返回时设置一个flag变量，

标志是int还是function，以便确定后面的内容。

2.bind the var to the function

--为方便起见，我根据assign的结果，

比如函数还是int，

选择去掉Var x End，并弄一个Function Dec，

还是保留Var x End。

每次assign都选择一次

int依旧保留var，

赋值的是function则再次重新function declaration

改完之后直接弄到father的sons中的对应次序。

3.Print--maybe should be considerded?

4.Return->int or function

5.Assign function to var

--can be replaced by the Function Declaration

[5-20-23:07]保存了prj1的代码，以备后患。

[5-21-00:09]发现，只能通过expr判断出int还是fun，

因为，print、apply、return都只能通过后面表达式的返回值才能够判断。

[5-21-15:35]return，assign的时候，

将返回的函数的指针，赋值到对应节点中。

return返回函数只在apply的时候用到，

就赋值到apply这个节点的pointToFun。

assign

[5-21-15:46]决定不用pointToFun，而是直接用number。

采用强制类型转换。

[5-21-16:13]再返回函数的时候，应该返回到callfunction的地方。

[5-21-16:15]return 时，无论VarName，还是Apply，都需要构造。

而且要注意换名字。

[5-21-16:18]当expr中VarName并且vartype为Fun时，

有Assign f h 的情况，

有Return h 的情况。

都需要构造。不如直接在expr-VarName-Fun时直接create。

另外，return apply返回函数时也是return，

所以expr-Apply-Fun时也直接creat

[5-21-16:33]检查一下赋值，call，apply的时候参数等等的intfun有没有一起赋值

[5-21-17:03]想起来Print 的 VarName并解决。

[5-22-14:54]发现根源的需要createFun的地方，只有return f的时候。以及assign f的时候

[5-22-15:24]有两个地方要改名字。

一个是assign a f 的时候的函数名，

只有assign的时候才知道最后的函数名。

还有一个是apply的时候的参数名

[5-22-15:53]发现，createFun的时候，参数fun是在apply或者

[5-22-16:27]所以最后考虑的两种情况

1.assign x f，

这时根据

assign->sons.back()->subtype==VarName

以及

assign->sons.back()->vartype==Fun

2.return t，包括assign x Apply f Argus End

一个特例：

apply apply x Argus End Argus 1 End

但是语法通不过，所以不考虑。

这时根据

Apply->father->sons.front()得到赋予的函数名，即x

[5-22-16:58]完成了上述情况1的处理，还差2

[5-22-17:12]弄一下参数是函数的情况

Function f Paras g h

Begin

Function t Paras i

Begin

Call g Argus End

return Apply t Argus

End

Return t

End

Assign x Apply

//想找一个使用函数作为参数，而不在函数内部使用apply或者call该函数参数的情况但是没想到。

[5-22-17:25]再弄情况2

[5-22-21:26]跑了一下测试1，调试之后通过了

[5-22-23:41]又想到一个例子。关于判断是否参数

[5-23-00:48]情况1跑完。

[5-23-01:04]测试完argu替换

[5-23-01:31]想到一个参数为函数，将其赋值给外部变量的例子。发现bug

[5-23-01:46]解决，正在弄return t的问题

[5-23-02:33]写完prjt2