变量

变量都是由指针实现。符号表中的每一个变量都是一个键值对<string name, Variable \* value>。其中符号表中的每一个项目，name代表这个变量的变量名，variable是一个指针，表示这个变量真正的内存地址。

运算：

Js中的一句a = b + c 实际上就对应着

Left = x ? b : 1;

Variable \*tmp = opAdditive(hash[b]->value, PLUS, hash[c]->value)

opAssign(hash[a]->value, ASSIGN, tmp);

在计算expression的时候，每一次计算都会返回一个中间对象，在使用完中间对象之后就会调用release将中间对象的retain减少1，如果这个中间对象被其他变量引用，那么release之后该变量不会被释放，否者由于减少后retain = 0，那么该变量就会被释放。

Javascript中的每一个变量在解释器中都是在堆中new出来的一个变量，使用中的就只有指针。

内存使用懒回收方式维护，每当new一个对象的时候，一个GC counter就会增加1，并且该对象的\_retain值置为0，并被丢到垃圾堆。当这个对象使用opAssign函数被赋值的时候，\_retain增加1，如果原先在垃圾堆中，那么就从垃圾堆中弹出。

如果当又一次new的时候，发现GC counter >= MAXN，就会触发垃圾回收机制。这时候会将所有垃圾堆中的对象全部delete掉。

其中，opAssign被定义为opAssign(Variable \*&value, TokenType, const Variable tmp);

参数中有带有&符号的，表示这个值会被修改，并被赋予一个在堆中new出来的一个对象。并且内部维护了对原来对象的释放操作。

Javascript中的一个临时变量就对应着c++中的一个值传递的variable类型。在计算过程中时时产生，在表达式计算结束的时候c++会释放。

Javascript中的一个实际变量就对应着c++中从堆中new出来的类型。当每一个javascript的变量名被赋值的时候，都会new出一个对象来，并将

Hash[a].value = op

Hash[name].value + hash[name]