**实验报告**

1. **实验目标**

在语法实验的基础上，修改语义动作，输出SysY代码对应的x86 AT&T汇编代码。

1. **实现功能**

能按要求翻译出汇编代码（并进行类型检查），但是存在部分缺陷：

1. 未进行寄存器等的优化
2. 定义数组时只能使用常数来定义数组的维度，不能使用变量或常量。比如样例中的const int N = 10; int a[N]; 就不能编译通过，需要改成int a[10];

（但调用数组元素时可以令下标为变量）

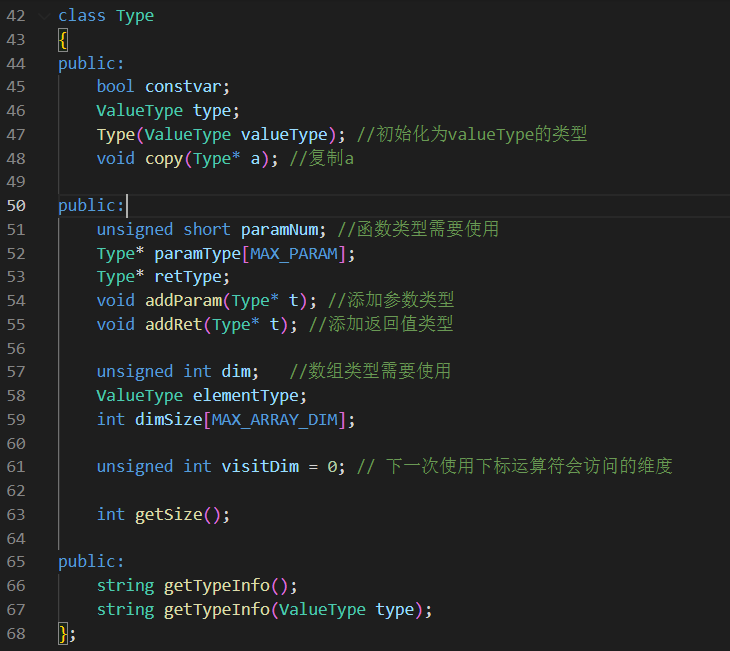
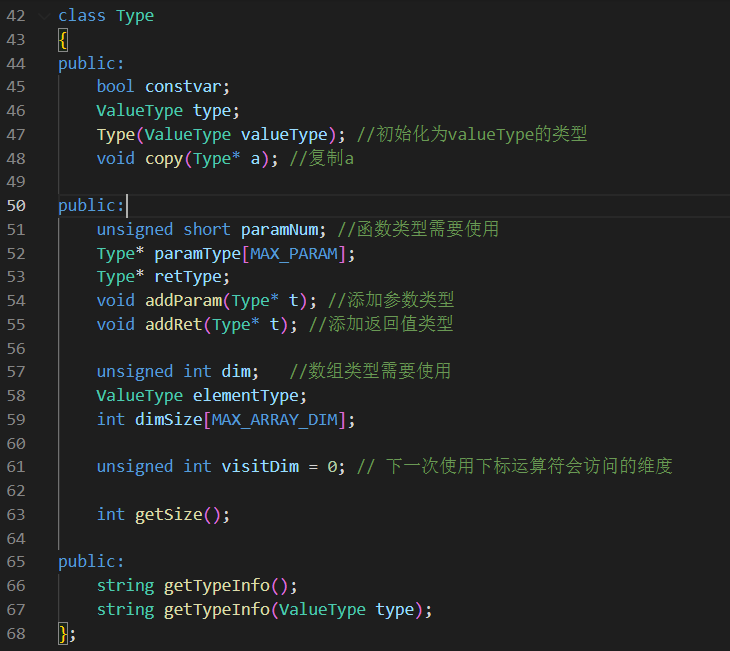
1. 没有保证16对齐
2. **具体实现**
3. **词法和语法部分**

这两部分与之前相比改动不大，只是由于改用了c++的语法，所以某些函数的形式发生了改变。另外为了配合汇编代码生成，对语法规则也进行了适量修改。

1. **语义规则**

利用之前语法实验时构建的语法树，我选择的实现方式是对构建好的语法树进行类型检查，然后将其翻译为汇编代码。(在“test.cpp”中)

（1）Type类来指示类型，包括int、数组、函数等等，并且为数组和函数类型设置了相应的成员变量。



成员函数中，Type(ValueType valueType)和copy函数进行初始化，getTypeInfo()返回相应的类型信息，addParam为函数类型添加参数的类型，addRet为函数类型添加返回值的类型，getSize则返回该类型所占空间大小。

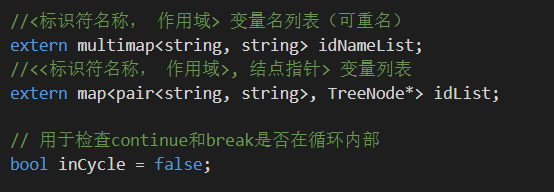
（2）TreeNode类，语法树节点，存储相应类型信息，以及运算符、表达式等的类型等等





初始化函数以及addChild和addBrother函数都与语法实验中相似，进行初始化和子节点构建。getLength返回子节点代表的链的长度，getVal返回为常数的节点的值。typeCheck进行类型检查。genCode将语法树翻译成汇编代码，gen\_var\_list、gen\_str、new\_list、get\_list、getVarPos辅助进行汇编代码生成（详细生成过程之后介绍）。

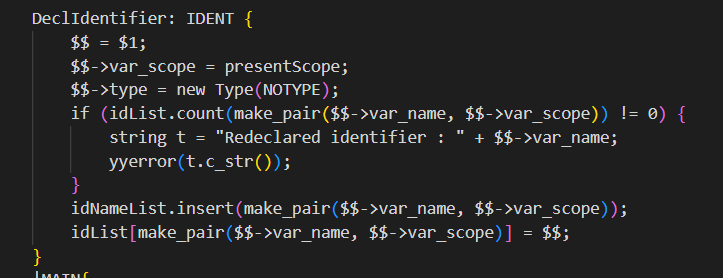
（3）首先设置全局的变量列表



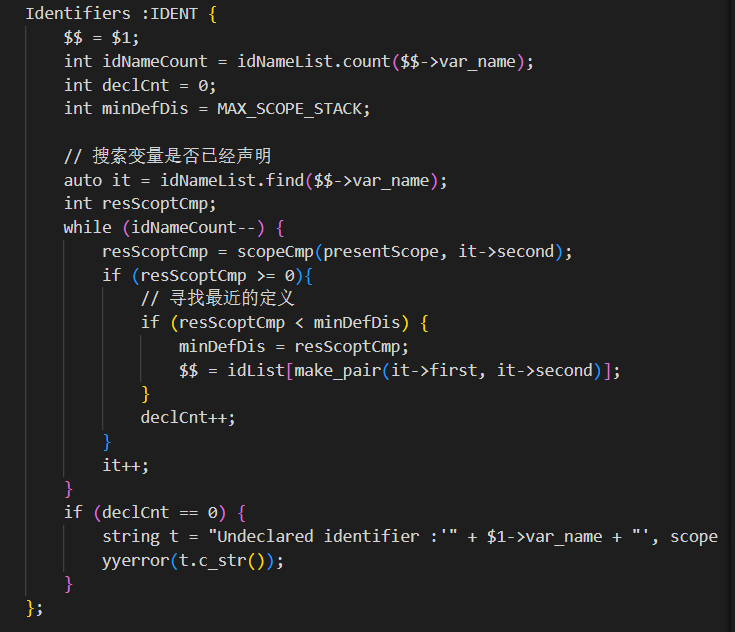
将YYSTYPE设置为TreeNode\*类型，在语义动作中构建语法树。

大多数语法规则的语义动作都是利用TreeNode(int num\_lines, NodeType type)和addChild来为待规约元素建立结点，并添加规约式中的非终结符结点作为子节点，此外还要根据规约式的性质（函数声明、表达式、语句等）来标记对应节点的Type类型。

但是对于变量的声明（Decl），要在识别出对应的标识符时检查其是否已经在与当前相交的作用域内被声明，并将未声明的标识符添加到变量列表中，比如：



而对于变量的调用，也需要在识别出对应标识符时检查变量列表，看是否有可用作用域范围内的同名变量，并且如果存在的话需要选最近的定义，从变量列表中读取出，比如：

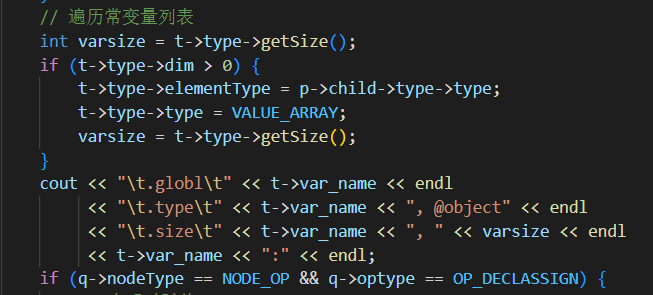


由于语法规则众多，其他语义动作不再一一说明。

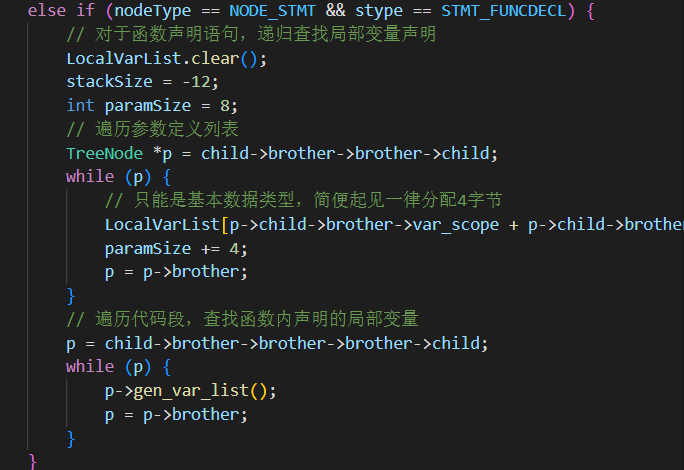
1. **汇编代码生成**

（1）gen\_var\_list()函数，生成对应区域变量列表

在根节点，即整个程序处，只处理全局变量的声明，用如下方式输出汇编代码

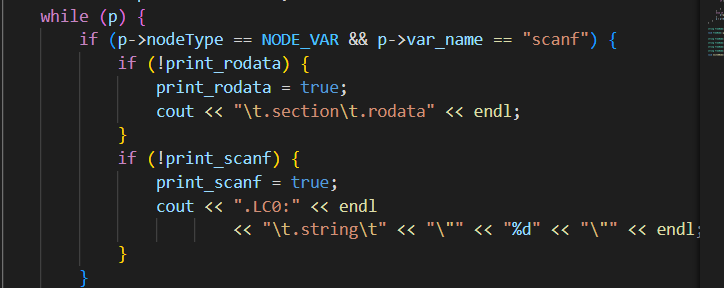


而对于局部变量，并不需要直接输出汇编代码，只需要计算所占空间和每个变量的相对位置即可。对函数声明语句，需要递归查找局部变量的声明，分配空间；对于局部变量声明语句，则需要根据其类型判断分配空间的大小



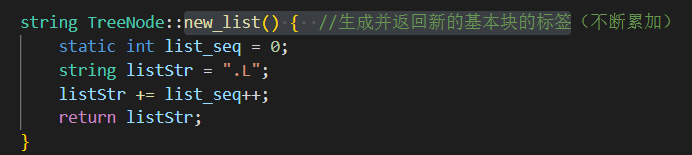
（2）gen\_str()函数，检测是否出现scanf和printf，输出对应模式串

在scanf和printf被调用的情况下，输出其对应的模式串



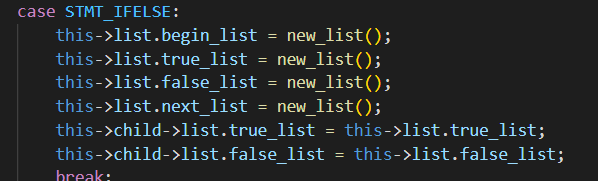
（3）new\_list()函数，生成并返回新的基本块的标签

用静态变量统计已有标签的个数，生成新的标签



（4）get\_list() 函数，分配并处理标签号

利用课上学过的知识，对IF-ELSE、WHILE语句及布尔表达式等的true-list和false-list进行处理，连接同样的list，比如对IF-ELSE语句



（5）getVarPos(TreeNode\* p) 函数，返回访问对应变量的地址

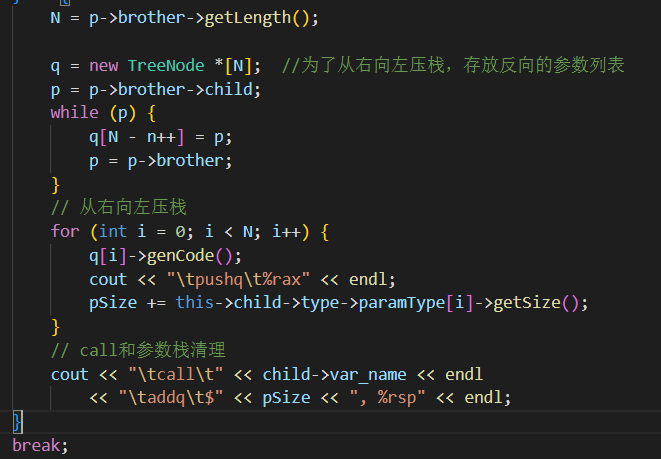
全局变量直接返回其标签地址，局部变量则利用局部变量列表求得它的相对地址。



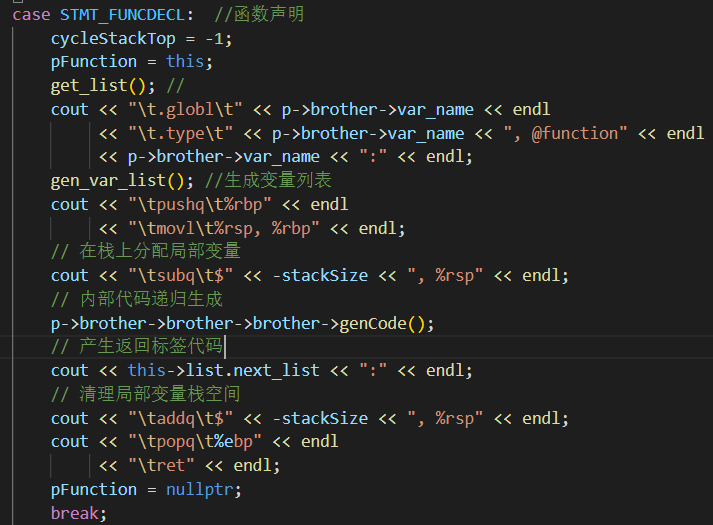
（6）genCode()函数，生成汇编代码

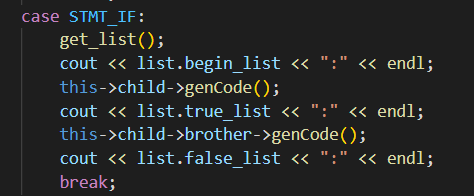
运用以上函数，对函数调用、STMT语句、Exp表达式、运算操作等不同的结点分别递归处理。（以下通过举少数例子来说明，详见“test.cpp”）

比如函数调用要先将参数从右向左压栈，再call调用，如下：

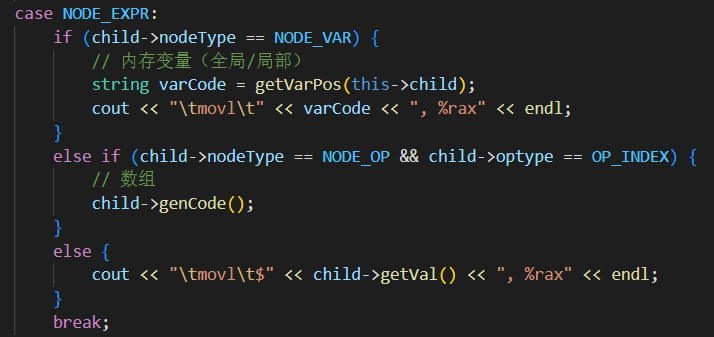


语句也要分类处理，如函数声明和IF语句，都要递归处理内部代码

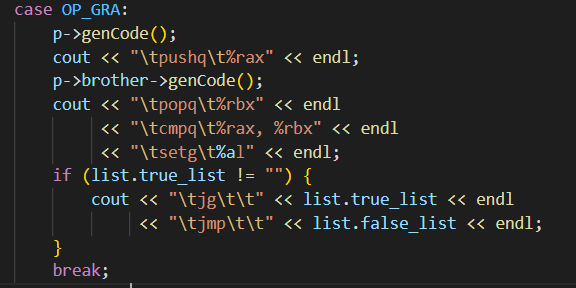


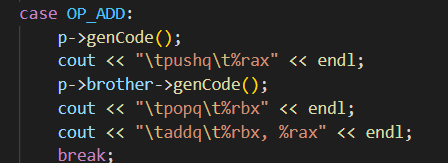


对表达式，令其值一律存放在%rax寄存器中返回，以方便运算操作时处理



对运算操作，分类处理，比如对大于关系和求和运算





1. **使用方法及样例展示**
2. 使用方法（以test.sy为例）

flex sysy.l

yacc -d yacc.y

g++ -c lex.yy.c

g++ -c y.tab.c

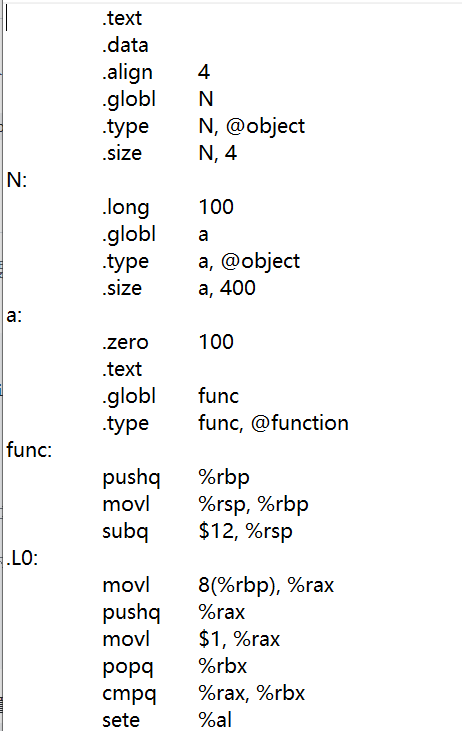
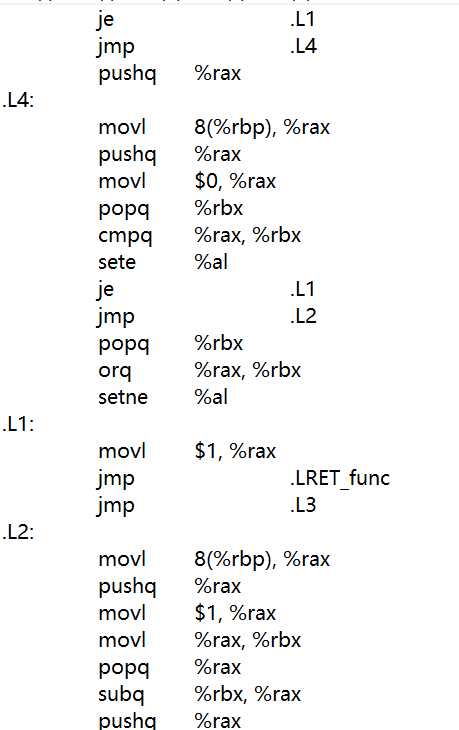
g++ -c test.cpp

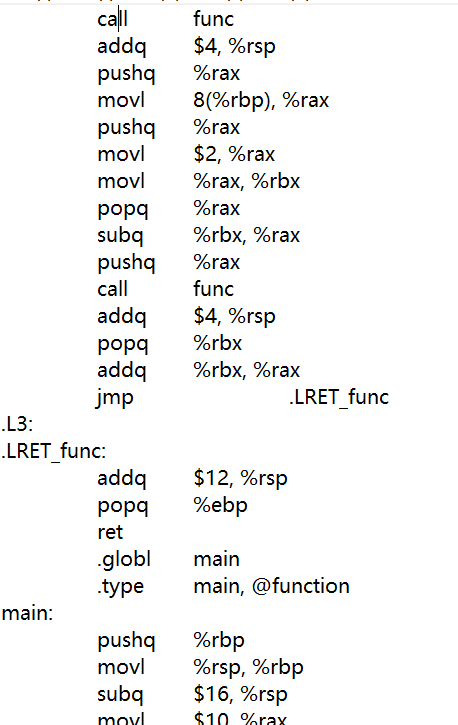
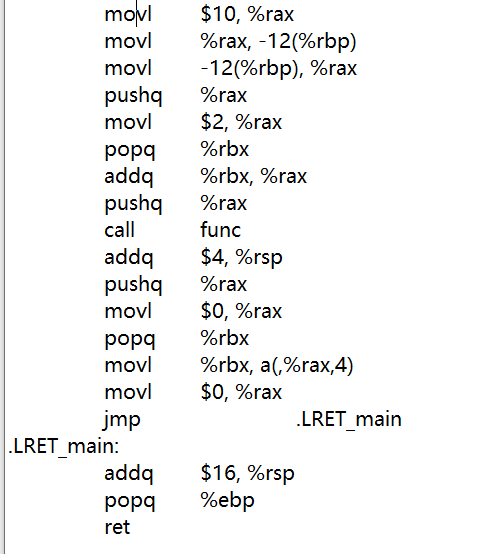
g++ lex.yy.o y.tab.o test.o -o main

./main < test.sy > test.out

1. 样例展示

应用于test.sy，得到输出如下（存放在test.out）中

另外四个样例在此不再展示，存放在对应的out文件中