西北工业大学

《编译原理》语法分析实验

抽象语法树

学　　院：　　软件学院

学　　号：　　2018303081

姓　　名：　　马泽红

专　　业：　　软件工程

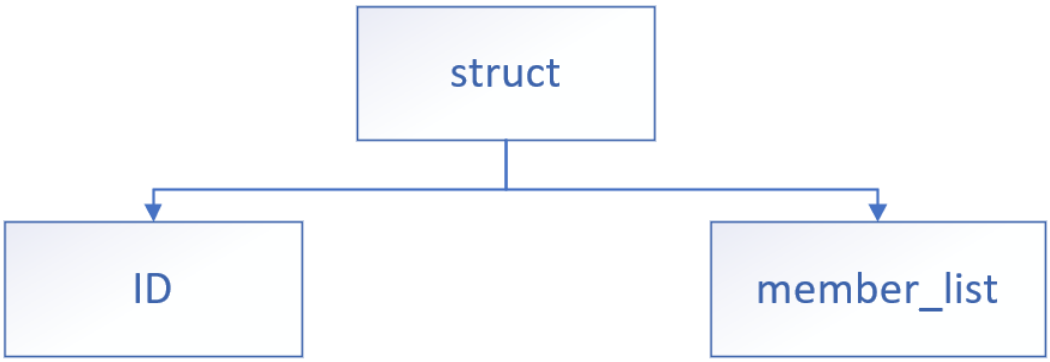
西北工业大学

2021 年 5 月

目录

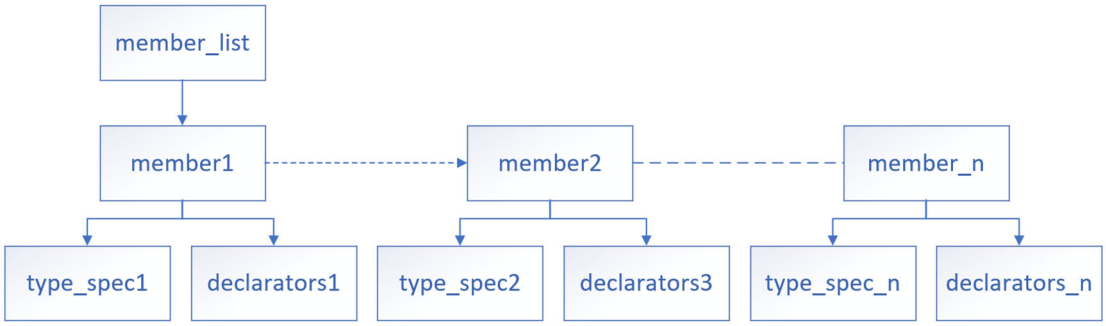
抽象语法树

1.struct_type->“struct” ID “{” member_list “}”



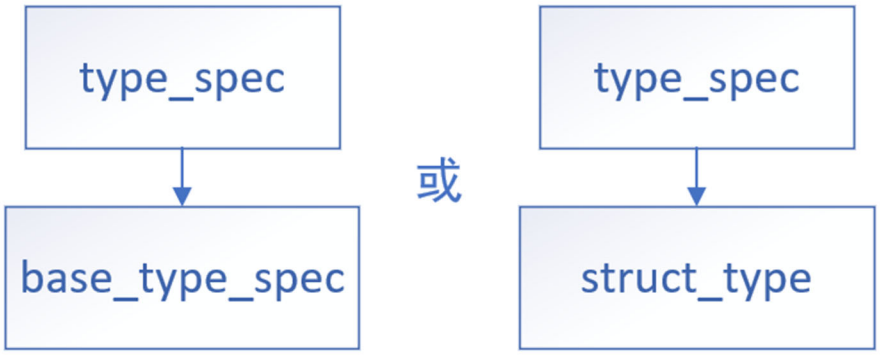
用 struct 作为根节点来标识此抽象语法树对应 struct_type 产生式，同时儿子节点省略掉左右中括号{}。

2.member_list-> { type_spec declarators “;” }



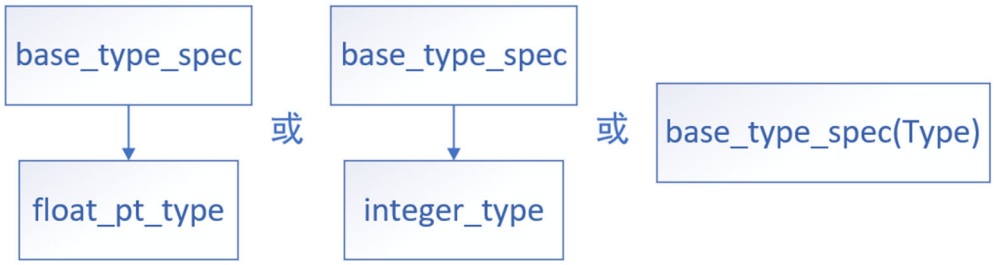
用 member_list 作为根节点来标识此抽象语法树对应 member_list 产生式；member1 为 member_list 儿子节点，代表 type_spec declarators 对应的子树的根节点；member2 是 member1 的兄弟节点，member_n 是 member_n-1 的兄弟节点，以此类推；type_spec1 和 declarators1 是 member1 的儿子节点，分别代表类型和变量声明的标识符，其他 member 以此类推。

3.type_spec -> base_type_spec | struct_type



type_spec 可以推导出 base_type_spec 或 struct_type 两种子类型。

4.base_type_spec->floating_pt_type|integer_type|“char”|“boolean”

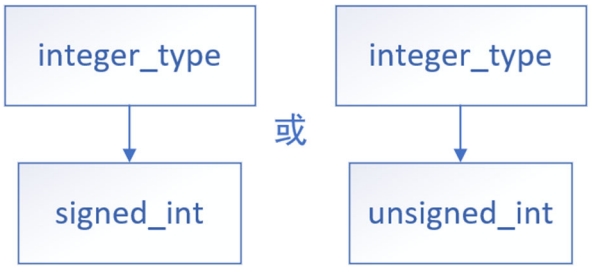


base_type_spec 可以推导出 float_pt_type 或 integer_type 或 char 或 boolean 四种子类型，当推导出 char 或 boolean 时用 base_type_spec(Type)表示，其中 Type 取值为 char 或 boolean。

5. floating_pt_type -> “float” | “double” | “long double”

floating_pt_type 可以推导出 float 或 double 或 long double 三种子类型,故 Type 取值为 float 或 double 或 long double。

6. integer_type -> signed_int | unsigned_int



integer_type 可以推导出 signed_int 或 unsigned_int 两种子类型。

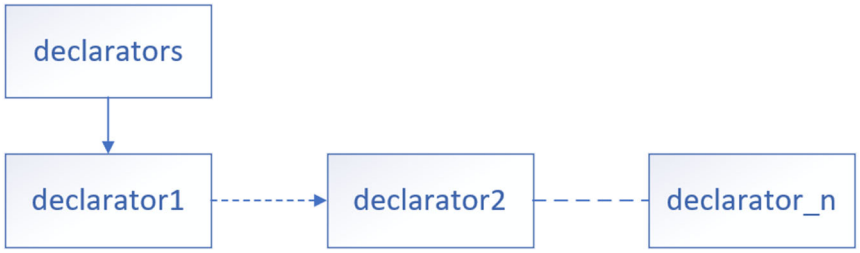
7.signed_int->(“short”|“int16”)|(“long”|“int32”)|(“long” “long”|“int64”)|“int8”

Type 取值为 short,int16,long,int32,long long,int64,int8。

8.unsigned_int->(“unsigned”“short”|“unit16”)|(“unsigned”“long”|“unit32”)|(“unsigned” “long” “long”| “unit64”)| “unit8”

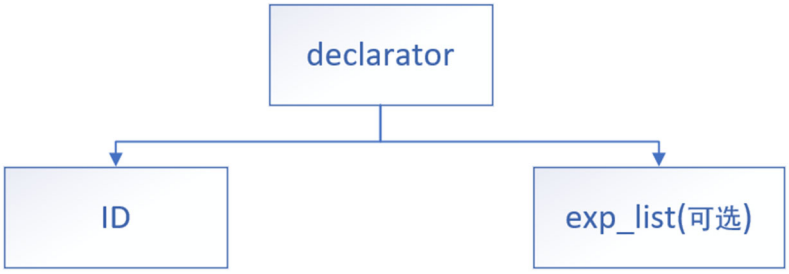
Type 取值为 unsigned short,uint16,unsigned,long,uint32,unsigned long long,uint64,uint8。

9. declarators -> declarator {“,” declarator }



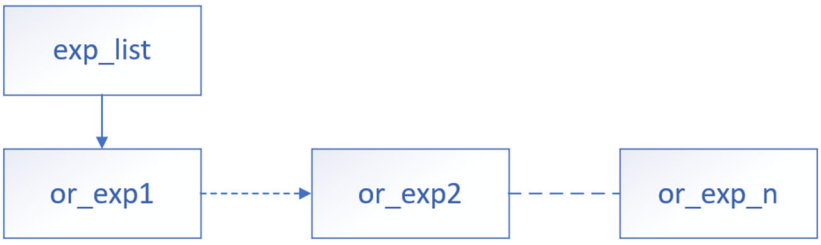
declarator1 是 declarators 的子节点，declarator2 是 declarator1 的兄弟节点，declarator_n 是 declarator_n-1 的兄弟节点。另外，需要注意，declarator 可以没有兄弟节点。

10. declarator -> ID [exp_list]



declarator 的子节点有 ID 和 exp_list，其中 exp_list 是可选的，有或没有都是合法的。

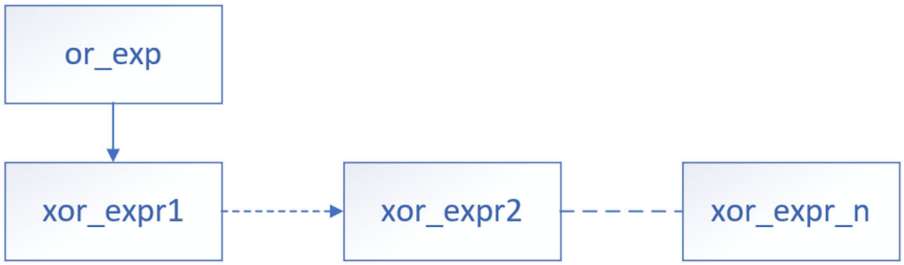
11.exp_list->“[”or_expr{“,”or_expr}“]”



or_exp1 是 exp_list 的子节点，or_exp2 是 or_exp1 的兄弟节点，or_exp_n 是

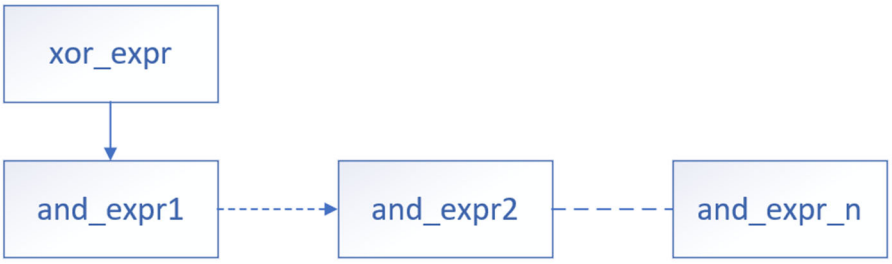
or_exp_n-1 的兄弟节点。另外，需要注意，or_exp 可以没有兄弟节点。左右中括号及“，”由于省略无歧义，故均被省略。

12.or_expr->xor_expr{“|”xor_expr}



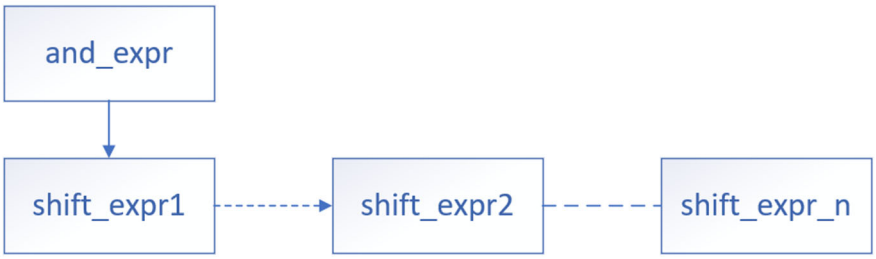
xor_expr1 是 or_exp 的子节点，xor_expr2 是 xor_expr1 的兄弟节点，xor_expr_n 是 xor_expr_n-1 的兄弟节点。另外，需要注意，xor_expr 可以没有兄弟节点。同时，省略掉“|”无歧义。

13. xor_expr -> and_expr {“^” and_expr }



and_expr1 是 xor_expr 的子节点，and_expr2 是 and_expr1 的兄弟节点，and_expr_n 是 and_expr_n-1 的兄弟节点。另外，需要注意，and_expr 可以没有兄弟节点。

14.and_expr->shift_expr{“&”shift_expr}



shift_expr1 是 and_expr 的子节点，shift_expr2 是 shift_expr1 的兄弟节点，shift_expr_n 是 shift_expr_n-1 的兄弟节点。另外，需要注意，shift_expr 可以没有兄弟节点。同时，由于省去“&”无歧义，故省略。

15.shift_expr->add_expr{(“>>”|“<<”)add_expr}

add_expr1 是 shift_expr 的子节点，并且其对应的操作符 Opk 为 null，add_expr2 是 add_expr1 的兄弟节点，其中 Opk 取值为“<<”或“>>”add_expr_n 是 add_expr_n-1 的兄弟节点。另外，需要注意，add_expr1 可以没有兄弟节点。

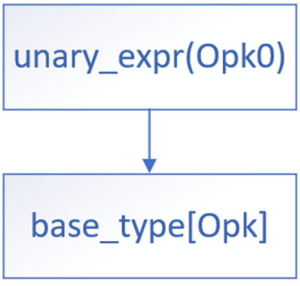
16.add_expr->mult_expr{(“+”|“-”)mult_expr}

mult_expr1 是 add_expr 的子节点，并且其对应的操作符 Opk 为 null，mult_expr2 是 mult_expr1 的兄弟节点，其中 Opk 取值为“+”或“-”。mult_expr_n 是 mult_expr_n-1 的兄弟节点。另外，需要注意，mult_expr1 可以没有兄弟节点。

17.mult_expr->unary_expr{(“*”|“/”|“%”)unary_expr}

unary_expr1 是 mult_expr 的子节点，并且其对应的操作符 Opk 为 null，unary_expr2 是 unary_expr1 的兄弟节点，其中 Opk 取值为“*”或“/”或“%”，unary_expr_n 是 unary_expr_n-1 的兄弟节点。另外，需要注意，unary_expr1 可以没有兄弟节点。注意：Opk0 为 mult_expr 对应的操作符“+”或“-”。

18.unary_expr->[“-”|“+”|“~”](INTEGER|STRING|BOOLEAN)



base_type 是 unary_expr 的子节点，其可取“INTEGER”或“STRING”或“BOOLEAN”。并且其对应的操作符 Opk 为可选，若没有则取 null，若有，则 Opk 取“-”或“+”或“~”。注意：Opk0 为 unary_expr 对应的操作符“*”或“/”或“%”