**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

[1.struct\_type->“struct” ID “{” member\_list “}” 2](#_Toc71759366)

[2.member\_list-> { type\_spec declarators “;” } 2](#_Toc71759367)

[3.type\_spec -> base\_type\_spec | struct\_type 3](#_Toc71759368)

[4.base\_type\_spec->floating\_pt\_type|integer\_type|“char”|“boolean” 3](#_Toc71759369)

[5. floating\_pt\_type -> “float” | “double” | “long double” 3](#_Toc71759370)

[6. integer\_type -> signed\_int | unsigned\_int 4](#_Toc71759371)

[7.signed\_int->(“short”|“int16”)|(“long”|“int32”)|(“long” “long”|“int64”)|“int8” 4](#_Toc71759372)

[8.unsigned\_int->(“unsigned”“short”|“unit16”)|(“unsigned”“long”|“unit32”)|(“unsigned”“long” “long”| “unit64”)| “unit8” 4](#_Toc71759373)

[9. declarators -> declarator {“,” declarator } 5](#_Toc71759374)

[10. declarator -> ID [ exp\_list ] 5](#_Toc71759375)

[11.exp\_list->“[”or\_expr{“,”or\_expr}“]” 5](#_Toc71759376)

[12.or\_expr->xor\_expr{“|”xor\_expr} 6](#_Toc71759377)

[13. xor\_expr -> and\_expr {“^” and\_expr } 6](#_Toc71759378)

[14.and\_expr->shift\_expr{“&”shift\_expr} 7](#_Toc71759379)

[15.shift\_expr->add\_expr{(“>>”|“<<”)add\_expr} 7](#_Toc71759380)

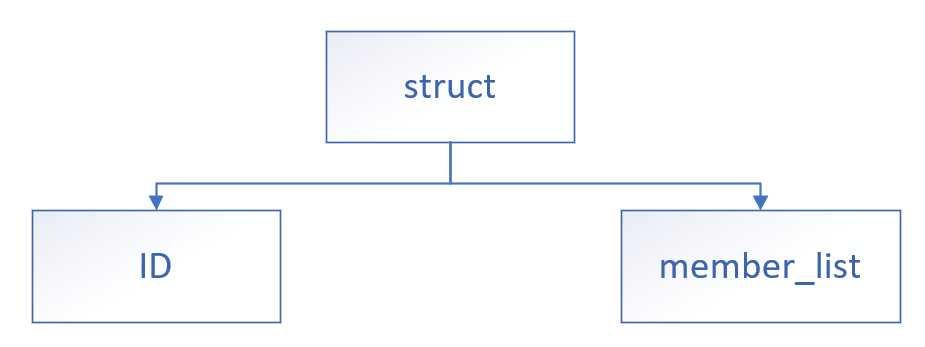
[16.add\_expr->mult\_expr{(“+”|“-”)mult\_expr} 7](#_Toc71759381)

[17.mult\_expr->unary\_expr{(“\*”|“/”|“%”)unary\_expr} 8](#_Toc71759382)

[18.unary\_expr->[“-”|“+”|“~”](INTEGER|STRING|BOOLEAN) 8](#_Toc71759383)

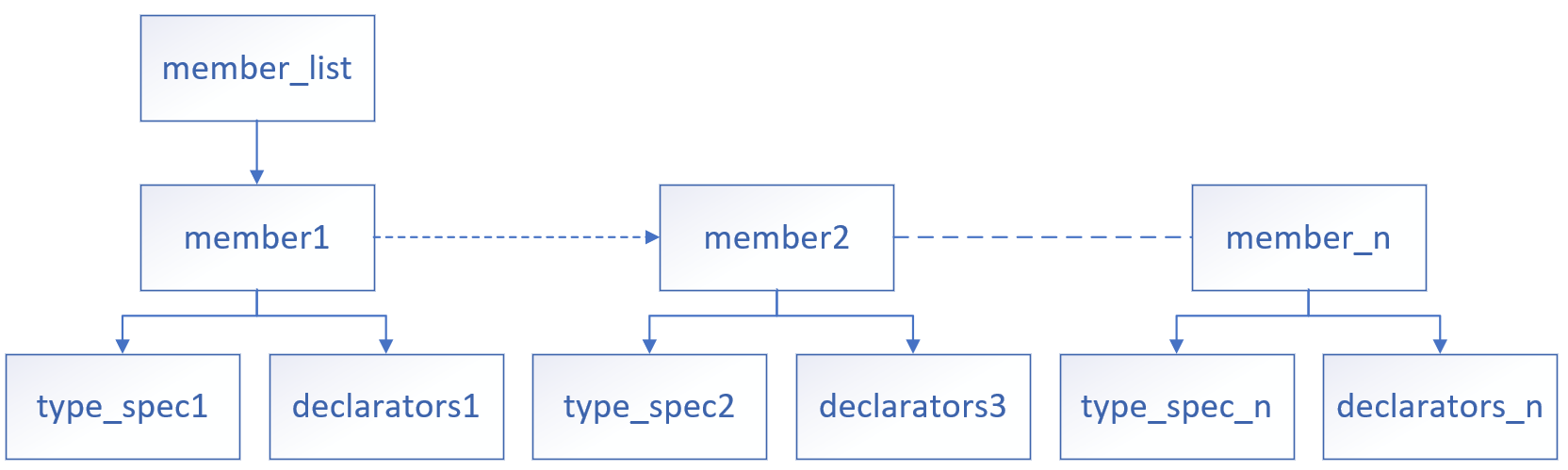
**抽象语法树**

# 1.struct\_type->“struct” ID “{” member\_list “}”



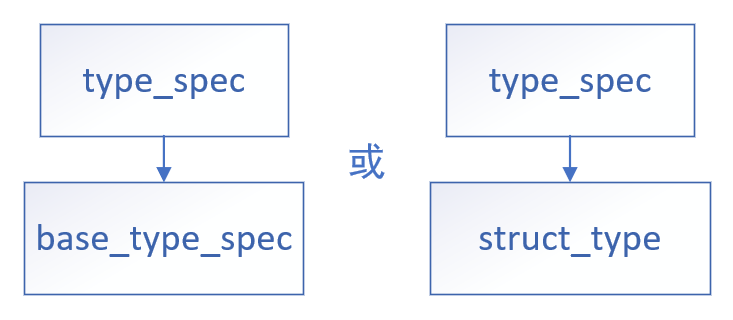
用struct作为根节点来标识此抽象语法树对应struct\_type产生式，同时儿子节点省略掉左右中括号{}。

# 2.member\_list-> { type\_spec declarators “;” }



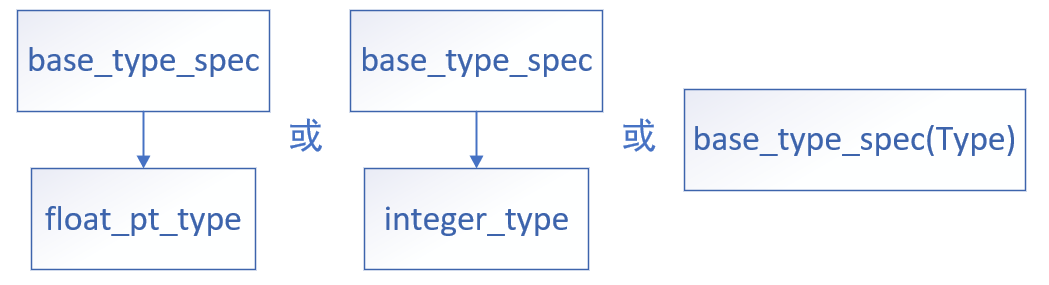
用member\_list作为根节点来标识此抽象语法树对应member\_list产生式；member1为member\_list儿子节点，代表type\_spec declarators对应的子树的根节点；member2是member1的兄弟节点，member\_n是member\_n-1的兄弟节点，以此类推；type\_spec1和declarators1是member1的儿子节点，分别代表类型和变量声明的标识符，其他member以此类推。

# 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”



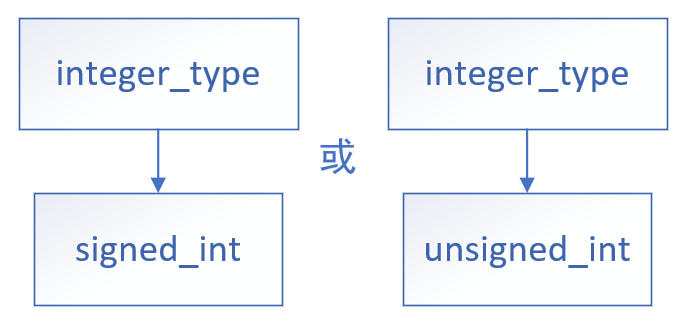
base\_type\_spec可以推导出float\_pt\_type或integer\_type或char或boolean四种子类型，当推导出char或boolean时用base\_type\_spec(Type)表示，其中Type取值为char或boolean。

# 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”



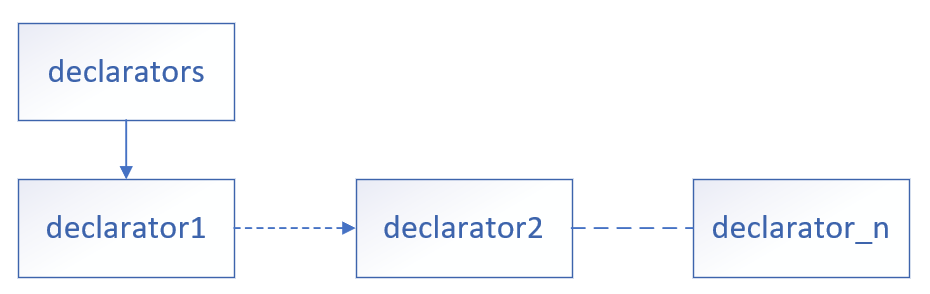
Type取值为short,int16,long,int32,long long,int64,int8。

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



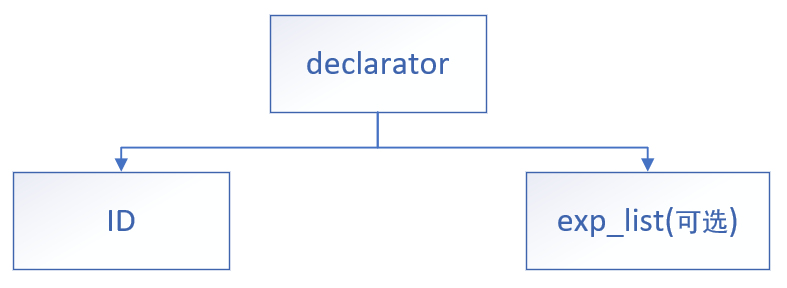
Type取值为unsigned short,uint16,unsigned,long,uint32,unsigned long long,uint64,uint8。

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



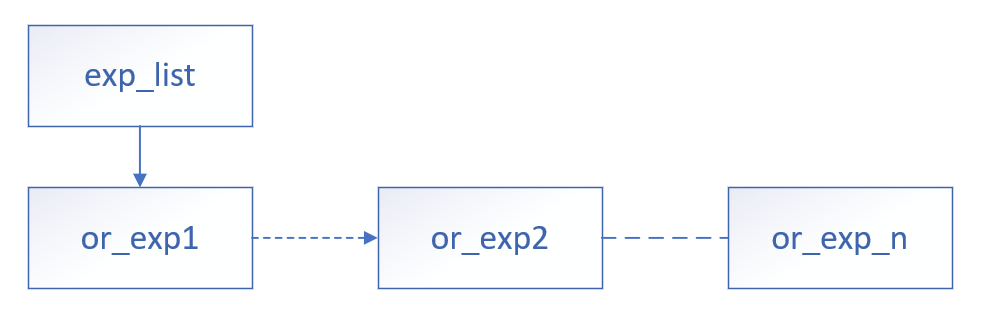
declarator1是declarators的子节点，declarator2是declarator1的兄弟节点，declarator\_n是declarator\_n-1的兄弟节点。另外，需要注意，declarator可以没有兄弟节点。

# 10. declarator -> ID [ exp\_list ]



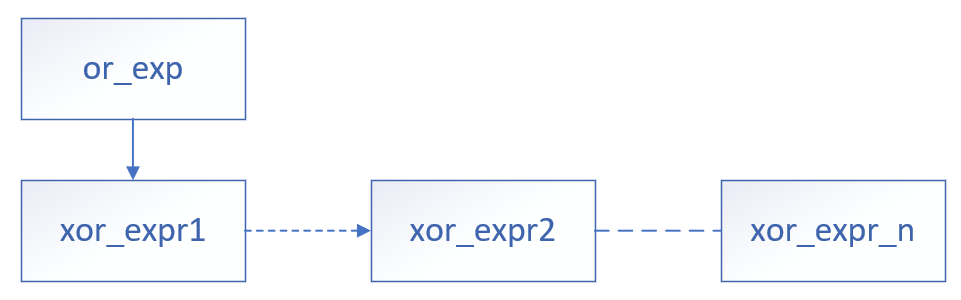
declarator的子节点有ID和exp\_list，其中exp\_list是可选的，有或没有都是合法的。

# 11.exp\_list->“[”or\_expr{“,”or\_expr}“]”



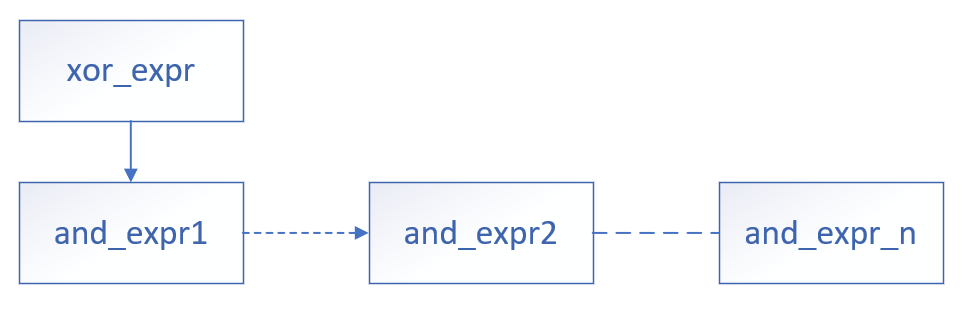
or\_exp1是exp\_list的子节点，or\_exp2是or\_exp1的兄弟节点，or\_exp\_n是or\_exp\_n-1的兄弟节点。另外，需要注意，or\_exp可以没有兄弟节点。左右中括号及“，”由于省略无歧义，故均被省略。

# 12.or\_expr->xor\_expr{“|”xor\_expr}



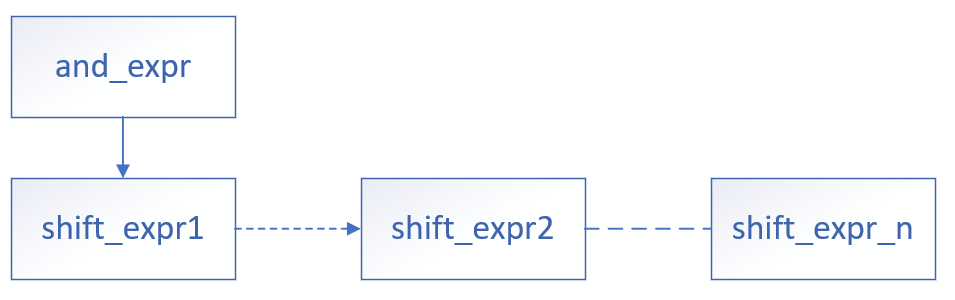
xor\_expr1是or\_exp的子节点，xor\_expr2是xor\_expr1的兄弟节点，xor\_expr\_n是xor\_expr\_n-1的兄弟节点。另外，需要注意，xor\_expr可以没有兄弟节点。同时，省略掉“|”无歧义。

# 13. xor\_expr -> and\_expr {“^” and\_expr }



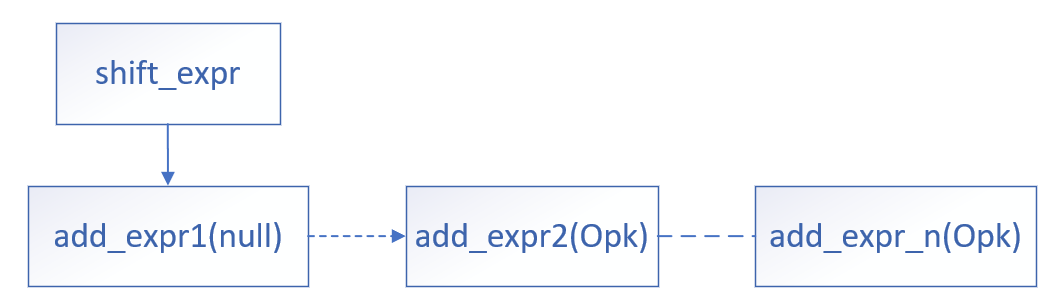
and\_expr1是xor\_expr的子节点，and\_expr2是and\_expr1的兄弟节点，and\_expr\_n是and\_expr\_n-1的兄弟节点。另外，需要注意，and\_expr可以没有兄弟节点。

# 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}



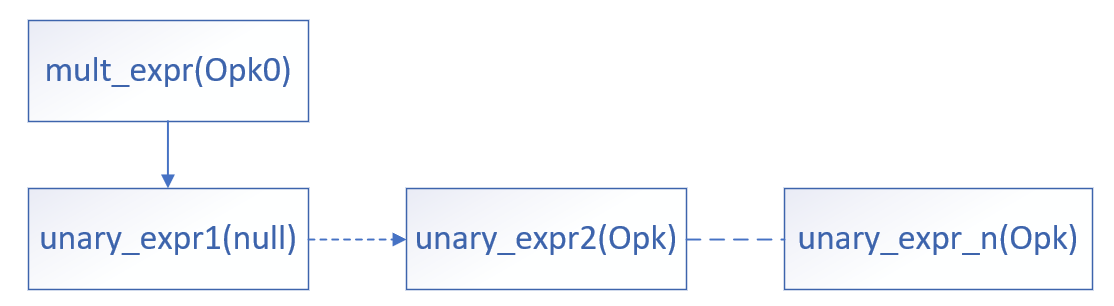
add\_expr1是shift\_expr的子节点，并且其对应的操作符Opk为null，add\_expr2是add\_expr1的兄弟节点，其中Opk取值为“<<”或“>>”add\_expr\_n是add\_expr\_n-1的兄弟节点。另外，需要注意，add\_expr1可以没有兄弟节点。

# 16.add\_expr->mult\_expr{(“+”|“-”)mult\_expr}



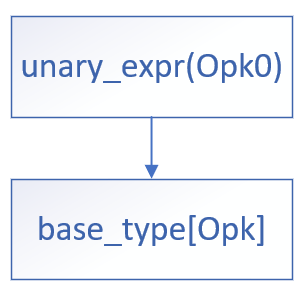
mult\_expr1是add\_expr的子节点，并且其对应的操作符Opk为null，mult\_expr2是mult\_expr1的兄弟节点，其中Opk取值为“+”或“-”。mult\_expr\_n是mult\_expr\_n-1的兄弟节点。另外，需要注意，mult\_expr1可以没有兄弟节点。

# 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“%”