

cilly_grammar

词法规范

标识符

```
identifier ::= identifier-nondigit
            | identifier identifier-nondigit
            | identifier digit;
```

其中, `identifier-nondigit` 为下划线, 小写英文字母或大写英文字母; `digit` 为数字 0 到 9.

数值常量

```
integer-const    ::= decimal-const
                  | octal-const
                  | hexadecimal-const;
decimal-const    ::= nonzero-digit
                  | decimal-const digit;
octal-const      ::= "0"
                  | octal-const octal-digit;
hexadecimal-const ::= hexadecimal-prefix hexadecimal-digit
                  | hexadecimal-const hexadecimal-digit;
hexadecimal-prefix ::= "0x" | "0X";
```

其中, `nonzero-digit` 为数字 1 到 9; `octal-digit` 为数字 0 到 7; `hexadecimal-digit` 为数字 0 到 9, 或大写/小写字母 a 到 f.

语法规范

```
CompUnit    ::= [CompUnit] (FuncDef | Decl);

Decl        ::= ValDecl | VarDecl;

ValDecl     ::= "val" IDENT ":" BType "=" InitVal ";";
VarDecl     ::= "var" IDENT ":" BType "=" InitVal ";";
InitVal     ::= Exp;

FuncDef     ::= FuncType IDENT "(" [FuncFParams] ")" ["->" BType] Block;
FuncFParams ::= FuncFParam {"," FuncFParam};
FuncFParam  ::= IDENT ":" BType;
FuncRParams ::= Exp {"," Exp};
BType       ::= "i32";

Block       ::= "{" {BlockItem} "}";
BlockItem   ::= Decl | Stmt;
Stmt        ::= LVal "=" Exp ";"
              | Block
              | [Exp] ";"
              | "return" [Exp] ";";
              | "if" "(" Exp ")" Stmt ["else" Stmt]
              | "while" "(" Exp ")" Stmt
```

```

        | "continue"
        | "break"
        | FuncDef;

LVal      ::= IDENT;

Exp        ::= LOrExp;
PrimaryExp ::= "(" Exp ")" | Number | LVal;
Number     ::= INT_CONST;
UnaryExp   ::= PrimaryExp
              | UnaryOp UnaryExp
              | IDENT "(" [FuncRParams] ")";
UnaryOp    ::= "+" | "-" | "!";
MulExp     ::= UnaryExp | MulExp ("*" | "/" | "%") UnaryExp;
AddExp     ::= MulExp | AddExp ("+" | "-") MulExp;
RelExp     ::= AddExp | RelExp ("<" | ">" | "<=" | ">=") AddExp;
EqExp      ::= RelExp | EqExp ("==" | "!=") RelExp;
LAndExp    ::= EqExp | LAndExp "&&" EqExp;
LOrExp     ::= LAndExp | LOrExp "||" LAndExp;

```

AST 样例

测试代码

```

fn fact(n: i32) -> i32 {
    if(n == 0) return 1;
    return n * fact(n - 1);
}

fn feb(n: i32) -> i32 {
    if(n < 2) {
        return 1;
    } else {
        return feb(n - 1) + feb(n - 2);
    }
}

fn while_test() -> i32 {
    var n: i32 = 0;
    while(n < 10) {
        print(n);
        n = n + 1;
    }
    return n;
}

fn add(a: i32, b: i32) -> i32 {
    return a + b;
}

fn main () {
    while_test();
    val n: i32 = getint();
    val res: i32 = fact(n);
}

```

```

    print(res);

    val m: i32 = getint();
    print(feb(m));
}

```

AST

```

CompUnit { globaldefs: [FuncDef(FuncDef { ident: "fact", btype: Some(I32), funcparams:
Some(FuncFParams { params: [FuncFParam { ident: "n", btype: I32 } ] }, block: Block { items:
[Stmt(If { condition: Exp { lor_exp: And(Eq(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))))), Eq,
Add(Mul(Unary(Pri(Number(0))))))) }, then_branch: Ret(Some(Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(Pri(Number(1))))))) }, else_branch: None }, Stmt(Ret(Some(Exp {
lor_exp: And(Eq(Rel(Add(Mul(Mul(Unary(Pri(LVal(LVal { ident: "n" }))), Mul, FuncCall { ident: "fact",
funcparams: Some(FuncRParams { exps: [Exp { lor_exp:
And(Eq(Rel(Add(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))), Sub, Unary(Pri(Number(1))))))) } } )
)))))) ) } } ), FuncDef(FuncDef { ident: "feb", btype: Some(I32), funcparams: Some(FuncFParams {
params: [FuncFParam { ident: "n", btype: I32 } ] }, block: Block { items: [Stmt(If { condition: Exp {
lor_exp: And(Eq(Rel(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))))), Lt,
Mul(Unary(Pri(Number(2))))))) }, then_branch: Block(Block { items: [Stmt(Ret(Some(Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(Pri(Number(1))))))) } ) ) ), else_branch: Some(Block(Block { items:
[Stmt(Ret(Some(Exp { lor_exp: And(Eq(Rel(Add(Add(Mul(Unary(FuncCall { ident: "feb", funcparams:
Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Add(Mul(Unary(Pri(LVal(LVal { ident: "n"
}))), Sub, Unary(Pri(Number(1))))))) } } ) ) ), Add, Unary(FuncCall { ident: "feb", funcparams:
Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Add(Mul(Unary(Pri(LVal(LVal { ident: "n"
}))), Sub, Unary(Pri(Number(2))))))) } } ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) )
)), FuncDef(FuncDef { ident: "while_test",
btype: Some(I32), funcparams: None, block: Block { items: [Decl(VarDecl(VarDecl { ident: "n",
btype: I32, initval: InitVal { exp: Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(Pri(Number(0))))))) } } )
}), Stmt(While { condition: Exp { lor_exp: And(Eq(Rel(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }
))))), Lt, Mul(Unary(Pri(Number(10))))))) }, loopbody: Block(Block { items: [Stmt(Exp(Some(Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "print", funcparams: Some(FuncRParams { exps: [Exp
{ lor_exp: And(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))))))) } } ) ) ) ) ) ) ) ) ) ) ) )
}), Stmt(Assign(LVal { ident: "n" }, Exp { lor_exp: And(Eq(Rel(Add(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }
))))), Add,
Unary(Pri(Number(1))))))) } ) ) ), Stmt(Ret(Some(Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))))))) } ) ) ), FuncDef(FuncDef { ident: "add",
btype: Some(I32), funcparams: Some(FuncFParams { params: [FuncFParam { ident: "a", btype: I32
}, FuncFParam { ident: "b", btype: I32 } ] }, block: Block { items: [Stmt(Ret(Some(Exp { lor_exp:
And(Eq(Rel(Add(Add(Mul(Unary(Pri(LVal(LVal { ident: "a" }))), Add, Unary(Pri(LVal(LVal { ident: "b"
}))))))) } ) ) ), FuncDef(FuncDef { ident: "main", btype: None, funcparams: None, block: Block {
items: [Stmt(Exp(Some(Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "print",
funcparams: Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall {
ident: "add", funcparams: Some(FuncRParams { exps: [Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(Pri(Number(100))))))) }, Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(Pri(Number(100))))))) } } ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) )
}), Stmt(Exp(Some(Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "while_test", funcparams: None
} ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) ) )
Decl(ValDecl(ValDecl { ident: "n", btype: I32, initval: InitVal { exp: Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "getint", funcparams: None } ) ) ) ) ) ) ) ) ) ) ) ) )
Decl(ValDecl(ValDecl { ident: "res", btype: I32, initval: InitVal { exp: Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "fact", funcparams: Some(FuncRParams { exps: [Exp {
lor_exp: And(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "n" }))))))) } } ) ) ) ) ) ) ) ) ) ) ) ) )
}), Stmt(Exp(Some(Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "print", funcparams:

```

```
Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal { ident: "res"
)))))))] } } ] })), Decl(ValDecl(ValDecl { ident: "m", btype: I32, initval: InitVal { exp: Exp { lor_exp:
And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "getint", funcrparams: None })))))] } })),
Stmt(Exp(Some(Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "print", funcrparams:
Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(FuncCall { ident: "feb",
funcrparams: Some(FuncRParams { exps: [Exp { lor_exp: And(Eq(Rel(Add(Mul(Unary(Pri(LVal(LVal {
ident: "m" )))))))] } } ] })))))] } } ] })))))] } } ] }
```

解释器结果

input

```
10
10
```

output

```
200
0
1
2
3
4
5
6
7
8
9
3628800
89
```

虚拟机字节码表

LoadConst(i32),	1	加载一个常数到栈顶。
LoadTrue,	2	加载True到栈顶。
LoadFalse,	3	加载False到栈顶。
LoadNull,	4	加载NULL到栈顶。
LoadGlobal(usize),	5	从全局变量中加载一个变量到栈顶。
StoreGlobal(usize),	6	将栈顶的值储存到全局变量表中。
BinOpAdd,	100	栈顶两个值相加。
BinOpSub,	101	栈顶两个值相减。
BinOpMul,	102	栈顶两个值相乘。
BinOpDiv,	103	栈顶两个值相除。
BinOpGt,	104	比较栈顶两个值，大于则结果为true。
BinOpGe,	105	比较栈顶两个值，大于等于则结果为true。
BinOpLt,	106	比较栈顶两个值，小于则结果为true。
BinOpLe,	107	比较栈顶两个值，小于等于则结果为true。
BinOpEq,	108	比较栈顶两个值，相等则结果为true。
BinOpNe,	109	比较栈顶两个值，不相等则结果为true。
BinOpOr,	110	栈顶两个值或。
BinOpAnd,	111	栈顶两个值与。
// 跳转的地址		
Jump(usize),	10	无条件跳转到指定位置。

JumpTrue(usize),	11	如果栈顶值为true, 跳转到指定位置。
JumpFalse(usize),	12	如果栈顶值为false, 跳转到指定位置。
PrintItem,	13	打印栈顶的值。
PrintNewline,	14	打印一个换行符。
GetInt,	15	输入一个整数
Pop,	16	弹出栈顶值
UniOpNot,	17	对栈顶的布尔值取反。
UniOpNeg,	18	对栈顶的值取负。
StorePC,	19	存储当前 PC。
LoadPC,	20	从 PC 栈中加载。
// dep, pos		
StoreVar(usize, usize),	21	将栈顶的值存储到局部变量表中。
// dep, pos		
LoadVar(usize, usize),	22	从局部变量表中加载一个变量到栈顶。
// args个数		
EnterScope(usize),	23	进入一个新的作用域。
LeaveScope,	24	离开当前作用域。
MakeClosure,	25	创建一个闭包。
// pc_addr, args个数		
Call(usize, usize),	26	调用一个函数。
Ret,	27	从当前函数返回。

测试样例生成的字节码

```

0      Call(58, 0)
1      EnterScope(0)
2      LoadVar(1, 0)
3      LoadConst(0)
4      BinOpEq
5      JumpFalse(8)
6      LoadConst(1)
7      Ret
8      LoadVar(1, 0)
9      LoadVar(1, 0)
10     LoadConst(1)
11     BinOpSub
12     Call(1, 1)
13     BinOpMul
14     Ret
15     LeaveScope
16     EnterScope(0)
17     LoadVar(1, 0)
18     LoadConst(2)
19     BinOpLt
20     JumpFalse(23)
21     LoadConst(1)
22     Ret
23     LoadVar(1, 0)
24     LoadConst(1)
25     BinOpSub
26     Call(16, 1)
27     LoadVar(1, 0)
28     LoadConst(2)
29     BinOpSub

```

```

30    Call(16, 1)
31    BinOpAdd
32    Ret
33    LeaveScope
34    EnterScope(0)
35    LoadConst(0)
36    StoreVar(0, 0)
37    LoadVar(0, 0)
38    LoadConst(10)
39    BinOpLt
40    JmpFalse(49)
41    LoadVar(0, 0)
42    PrintItem
43    PrintNewline
44    LoadVar(0, 0)
45    LoadConst(1)
46    BinOpAdd
47    StoreVar(0, 0)
48    Jmp(37)
49    LoadVar(0, 0)
50    Ret
51    LeaveScope
52    EnterScope(0)
53    LoadVar(1, 0)
54    LoadVar(1, 1)
55    BinOpAdd
56    Ret
57    LeaveScope
58    EnterScope(0)
59    Call(34, 0)
60    GetInt
61    StoreVar(0, 0)
62    LoadVar(0, 0)
63    Call(1, 1)
64    StoreVar(0, 1)
65    LoadVar(0, 1)
66    PrintItem
67    PrintNewline
68    GetInt
69    StoreVar(0, 2)
70    LoadVar(0, 2)
71    Call(16, 1)
72    PrintItem
73    PrintNewline
74    LeaveScope

```

```

26 58 0 23 0 22 1 0 1 0 108 12 8 1 1 27 22 1 0 22 1 0 1 1 101 26 1 1 102 27 24 23 0 22 1 0 1 2
106 12 23 1 1 27 22 1 0 1 1 101 26 16 1 22 1 0 1 2 101 26 16 1 100 27 24 23 0 1 0 21 0 0 22 0 0
1 10 106 12 49 22 0 0 13 14 22 0 0 1 1 100 21 0 0 10 37 22 0 0 27 24 23 0 22 1 0 22 1 1 100 27
24 23 0 26 34 0 15 21 0 0 22 0 0 26 1 1 21 0 1 22 0 1 13 14 15 21 0 2 22 0 2 26 16 1 13 14 24

```

虚拟机运行结果

input

```
10
10
```

output

```
0
1
2
3
4
5
6
7
8
9
10
3628800
10
89
```

过程截图：

解释器

```
● → cilly git:(main) X ./target/release/cilly --static ./res/test.cil
0
1
2
3
4
5
6
7
8
9
10
3628800
10
89
```

转化成字节码

```
● → cilly git:(main) X ./target/release/cilly --translate ./res/test.cil
./res/test.cby is created !
○ → cilly git:(main) X █

≡ test.cby ×
res > ≡ test.cby
1 26 58 0 23 0 22 1 0 1 0 108 12 8 1 1 27 22 1 0 22 1 0 1 1 101 26 1 1 102
```

虚拟机运行字节码

```
→ cilly git:(main) X ./target/release/cilly --vmrun ./res/test.cby
0
1
2
3
4
5
6
7
8
9
10
3628800
10
89
→ cilly git:(main) X
```