AST 打印规范

针对本次实验中新添加的5个语言特性，我们特别通过此打印规范文档，说明如何规范地将这些语法结点打印出来。 `TestCases/S1/result`目录下已经给出所有测例的标准输出，其中涉及新特性的部分均符合本文档之规范， 仅当你的输出与之完全一致时，才被视为正确。 对于已存在的各语法结点，请不要做任何修改。

本文假定一层缩进占4个空格。

对象复制语句

表达式`scopy(<identifier>, <expr>)`打印成

```
scopy
    <identifier>
    <expr>
```

如

```
scopy(p1,p);
```

打印成

```
scopy
    p1
    varref p
```

关键词sealed

形如`sealed class <identifier1> extends <identifier2> {<Field*>}`的语句打印成

```
sealed class <identifier1> <identifier2>
    <field*>
```

此处为简便，<field*>未展开

如

```
sealed class A extends B {}
```

打印为

```
sealed class A B
```

串行条件卫士语句

形如

```
if { <expr1> : <stmt1> |||
     <expr2> : <stmt2> |||
     ...
     <exprN> : <stmtN> }
```

的串行卫士语句，打印为

```
guarded
    guard
        <expr1>
        <stmt1>
    guard
        <expr2>
        <stmt2>
    ...
    guard
        <exprN>
        <stmtN>
```

注意在开始打印各分支之前，先打印标识符`guarded`，增加缩进后才开始依次打印各分支。 对于每一个分支，都要先打印块起始标记`guard`，然后增加缩进，并先打印其条件`<expr>`，再打印其执行语句`<stmt>`。

如

```
if { y < 10: y = y + 1; |||
     x < 10: x = x + 1; }
```

打印为

```
guarded
    guard
        les
            varref y
            intconst 10
        assign
            varref y
            add
                varref y
                intconst 1
    guard
        les
            varref x
```

```
            intconst 10
        assign
            varref x
            add
                varref x
                intconst 1
```

自动类型推导

形如`var <identifier>`的自动类型推导语句打印为

```
  var <identifier>
```

如

```
var x = 1;
```

打印为

```
assign
    var x
    intconst 1
```

一维数组相关语句

数组常量

数组常量`[<constant1>,<constant2>...<constantN>]`打印为

```
array const
    <constant1>
    <constant2>
    ...
    <constantN>
```

如

```
xs = [1,2,3];
```

打印为

```
assign
    varref xs
    array const
        intconst 1
        intconst 2
        intconst 3
```

数组初始化常量表达式

数组初始化常量表达式`<expr> %% <intconst>`打印为

```
array repeat
    <expr>
    <intconst>
```

如

```
xs = 1 %% 10;
```

打印为

```
assign
    varref xs
    array repeat
        intconst 1
        intconst 10
```

数组拼接表达式

数组拼接表达式`<expr> ++ <expr>`打印为

```
array concat
    <expr>
    <expr>
```

如

```
int[] ws;
int[] xs;
int[] arr;
arr = ws ++ xs;
```

打印为

```
vardef ws arrtype inttype
vardef xs arrtype inttype
vardef arr arrtype inttype
assign
    varref arr
    array concat
        varref ws
        varref xs
```

取子数组表达式

子数组表达式`<expr1> [<expr2> : <expr3>]`打印为

```
arrref
    <expr1>
    range
        <expr2>
        <expr3>
```

如

```
sub1 = xs[2:6];
```

打印为

```
assign
    varref sub1
    arrref
        varref xs
        range
            intconst 2
            intconst 6
```

数组下标动态访问表达式

形如`<expr1> [<expr2>] default <expr3>`打印为

```
arrref
    <expr1>
    <expr2>
    default
        <expr3>
```

如

```
z1 = xs[11] default 0;
```

打印为

```
assign
    varref z1
    arrref
        varref xs
        intconst 11
        default
            intconst 0
```

Python风格的数组 comprehension 表达式

形如`[<expr1> for <identifier> in <expr2> if <expr3> ]`打印为

```
array comp
    varbind <identifier>
    <expr2>
    <expr3>
    <expr1>
```

前用标志符varbind修饰，注意的顺序

如

```
[x*y for x in xs if x >= 5]
```

打印为

```
array comp
    varbind x
    varref xs
    geq
        varref x
        intconst 5
    mul
        varref x
        varref y
```

数组迭代语句

数组迭代语句`foreach ( var <identifier> in <expr1> while <expr2> ) <stmt>`打印为

```
foreach
    varbind <identifier> var
    <expr1>
    <expr2>
    <stmt>
```

如果将var替换为Type，那么打印中也要做相应的替换，如果whle 省略时，即该条件恒为真， 对应的打印为 `boolconst true`

如

```
foreach (var s in ss while s.length() < 10) {
    Print(s);
}
```

打印为

```
foreach
    varbind s var
    varref ss
    les
        call length
            varref s
        intconst 10
    stmtblock
        print
            varref s
```