Computer Science Department
Programming Methods Laboratory
(LAMP)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Compilation **winter semester 1999/2000**

Tutorial Slides

29.10.99

Programming Language Implementation

A Simple J0 Program

```
module M {                              // module
    int i;                              // global variable

    void foo() {                        // function
        int k;                          // local variable
        k = 0;
        i = k;
    }

    int inc(int x) {
        return x + i       ;
    }

    int j;
}
```


Programming Language Implementation

J0 Namespaces

A name is defined,

- if the declaration is in the same or an enclosing block and
- if the name is declared textually before accessing it

If there is more than one declaration in the enclosing blocks, then only the innermost declaration can be accessed.

```
module M {
    int i;

    void init() {
        int i;
        i = foo(1);
    }

    int foo(int n) {
        if (n == 0) {
            return i;
        }
        return foo(n - 1);
    }
}
```


J0 Grammar

```
ModuleDeclaration = "module" ident "{" { Declaration } "}".

Declaration = VarDeclaration | FunDeclaration.

VarDeclaration = Type ident ";".

Type = "int" | "string" | Type "[" "]".

FunDeclaration = ( Type | "void" ) ident "(" [ Formals ] ")"
                 Block.
```

```
Formals = Formal { "," Formal }.

Formal = Type ident.

Block = "{" { BlockStatement } "}".

BlockStatement = Statement | VarDeclaration.

Statement = FunCall | Assignment | Block |
            IfStatement | WhileStatement | ReturnStatement.

FunCall = ident "(" [ Expressions ] ")" ";".

Assignment = ident { "[" Expression "]" } "=" Expression ";".

IfStatement = "if" "(" Expression ")" Statement
              [ "else" Statement ].

WhileStatement = "while" "(" Expression ")" Statement.

ReturnStatement = "return" Expression ";".

Expression = AndExpression | Expression "|" AndExpression.

AndExpression = CmpExpression | AndExpression "&" CmpExpression.

CmpExpression = SumExpression [ CompOp SumExpression ].

CompOp = "==" | "!=" | "<" | ">" | "<=" | ">=".

SumExpression = Term | SumExpression SumOp Term.

SumOp = "+" | "-".

Term = [ "-" ] Factor | Term ProdOp [ "-" ] Factor.

ProdOp = "*" | "/" | "%"

Factor = ident [ "(" [ Expressions ] ") ] { "[" Expression "]" } |
         number | string | "(" Expression ")" | "!" Factor |
         "newarray" Type "(" Expression ")".

Expressions = Expression { "," Expression }.
```

J0 Types

J0 offers two built in primitive types

- `int`
- `string`

and one-dimensional arrays of any type; e.g.

```
int[]
string[][][]
```


Programming Language Implementation

J0 Arrays

- a variable declaration for an array doesn’t create the array itself
- constructor: newarray <elementtype>(<arraysize>) creates a new array of size <arraysize>
- the indices range from 0 to <arraysize> - 1

```
int[] xs;
xs = newarray int(8);

string[][] ts;
ts = newarray string[](4);
```

an array element can be accessed with the [...] operator e.g.

```
xs[0] = 1;
ts[1] = newarray string(xs[0] + 1);
ts[1][0] = "I’m a string literal"
```


Passing Parameters and Results

- call-by-value for values of type int and string
- call-by-reference for arrays

```
void setOne(int[] xs) {
    xs[0] = 1;
}

void setTwo(int x) {
    x = 2;
}

int main() {
    int[] ys;
    ys = newarray int(1);
    ys[0] = 0;
    setOne(ys);
    setTwo(ys[0]);
    return ys[0];
}
```


Short-cut evaluation

- generally, before executing a binary operation, both arguments are evaluated first
- for the | and the & operator, short-cut evaluation is performed

in expression e1 & e2, e2 is only evaluated,

    if e1 evaluates to non-zero

in expression e1 | e2, e2 is only evaluated,

    if e1 evaluates to zero

What is wrong?

```
int fak(n) {
    if (n = 0) {
        return 1
    } else
        return { n * fak(n - 1) };
}
```


Last modified: 29.10.1999, Matthias Zenger <matthias.zenger@epfl.ch>