

# Context Free Grammar of KPL

## The notation used in this grammar:

*Non-terminal Symbols: e.g. HeaderFile, Type, Expr, Statement*

*Terminal Symbols:*

*Keywords: e.g. if, while, int, endWhile*

*Tokens:*

*INTEGER: e.g. 42, 0x1234ABCD*

*DOUBLE: e.g. 3.1415, 6.022e23*

*CHAR: e.g. 'a', '\n'*

*STRING: e.g. "hello", "\t\n"*

*ID: e.g. x, myName, MAX\_SIZE*

*OPERATOR: e.g. <=, <, >, >=, !=, +, -, \*, etc.*

*Misc Punctuation: e.g. :, ,, ., =, (, ), ;*

*Punctuation that conflicts with meta-symbols: '{', '}', '[', ']', '|'*

*Comment Conventions:*

*-- through end-of-line*

*/\* through \*/*

*Meta-Symbols, used in describing the grammar:*

*Grammar Rule: -->*

*Example:*

*Type --> int*

*Repetition of zero-or-more: { }*

*Example:*

*StmtList --> { Statement }*

*Rules with Alternatives:*

*Example:*

*Statement --> IfStmt | AssignStmt*

*Example:*

*Statement --> IfStmt  
--> AssignStmt*

*Optional Material: [ ]*

*Example:*

*VarDecl --> Decl [ = Expr2 ]*

*One-or-more Occurrences: { }+*

*Example:*

*VarDecls --> var { VarDecl }+*

# Context Free Grammar of KPL

## Keywords

<code>alloc</code>	<code>fields</code>
<code>anyType</code>	<code>for</code>
<code>array</code>	<code>free</code>
<code>arraySize</code>	<code>function</code>
<code>asInteger</code>	<code>header</code>
<code>asPtrTo</code>	<code>if</code>
<code>behavior</code>	<code>implements</code>
<code>bool</code>	<code>infix</code>
<code>break</code>	<code>int</code>
<code>by</code>	<code>interface</code>
<code>case</code>	<code>isInstanceOf</code>
<code>catch</code>	<code>isKindOf</code>
<code>char</code>	<code>messages</code>
<code>class</code>	<code>method</code>
<code>code</code>	<code>methods</code>
<code>const</code>	<code>new</code>
<code>continue</code>	<code>null</code>
<code>debug</code>	<code>of</code>
<code>default</code>	<code>prefix</code>
<code>do</code>	<code>ptr</code>
<code>double</code>	<code>record</code>
<code>else</code>	<code>renaming</code>
<code>elseif</code>	<code>return</code>
<code>endBehavior</code>	<code>returns</code>
<code>endClass</code>	<code>self</code>
<code>endcode</code>	<code>sizeof</code>
<code>endFor</code>	<code>super</code>
<code>endFunction</code>	<code>superclass</code>
<code>endHeader</code>	<code>switch</code>
<code>endif</code>	<code>throw</code>
<code>endInterface</code>	<code>to</code>
<code>endMethod</code>	<code>true</code>
<code>endRecord</code>	<code>try</code>
<code>endSwitch</code>	<code>type</code>
<code>endTry</code>	<code>typeOfNull</code>
<code>endWhile</code>	<code>until</code>
<code>enum</code>	<code>uses</code>
<code>errors</code>	<code>var</code>
<code>extends</code>	<code>void</code>
<code>external</code>	<code>while</code>
<code>false</code>	

# Context Free Grammar of KPL

```
HeaderFile      --> header ID
                  [ Uses ]
                  { Constants |
                    Errors |
                    VarDecls |
                    Enum |
                    TypeDefs |
                    FunctionProtos |
                    Interface |
                    Class }
                  endHeader
CodeFile        --> code ID
                  { Constants |
                    Errors |
                    VarDecls |
                    Enum |
                    TypeDefs |
                    Function |
                    Interface |
                    Class |
                    Behavior }
                  endcode
Interface       --> interface ID [ TypeParms ]
                  [ extends TypeList ]
                  [ messages { MethProto }+ ]
                  endInterface
Class           --> class ID [ TypeParms ]
                  [ implements TypeList ]
                  [ superclass NamedType ]
                  [ fields { Decl }+ ]
                  [ methods { MethProto }+ ]
                  endClass
Behavior        --> behavior ID
                  { Method }
                  endBehavior
Uses            --> uses OtherPackage { , OtherPackage }
OtherPackage    --> ID      [ renaming Rename { , Rename } ]
                --> STRING [ renaming Rename { , Rename } ]
Rename          --> ID to ID
TypeParms       --> '[' ID : Type { , ID : Type } ']'
Constants       --> const { ID = Expr }+
Decl            --> ID { , ID } : Type
VarDecl         --> Decl [ = Expr2 ]
VarDecls        --> var { VarDecl }+
Errors          --> errors { ID ParmList }+
TypeDefs        --> type { ID = Type }+
Enum            --> enum ID [ = Expr ] { , ID }
IdList          --> ID { , ID }
ArgList         --> ( )
                --> ( Expr { , Expr } )
ParmList        --> ( )
                --> ( Decl { , Decl } )
```

# Context Free Grammar of KPL

```
FunProto      --> ID ParmList [ returns Type ]
FunctionProtos --> functions { [ external ] FunProto }+
Function      --> function ID ParmList [ returns Type ]
               [ VarDecls ]
               StmtList
               endFunction
NamelessFunction --> function ParmList [ returns Type ]
                  [ VarDecls ]
                  StmtList
                  endFunction
MethProto     --> ID ParmList [ returns Type ]
               --> infix OPERATOR ( ID : Type ) returns Type
               --> prefix OPERATOR ( ) returns Type
               --> { ID : ( ID : Type ) }+ [ returns Type ]
Method        --> method MethProto
               [ VarDecls ]
               StmtList
               endMethod
StmtList      --> { Statement }
Statement     --> if Expr StmtList
               { elseif Expr StmtList }
               [ else StmtList ]
               endif
               --> LValue = Expr
               --> ID ArgList
               --> Expr { ID : Expr }+
               --> Expr . ID ArgList
               --> while Expr
                   StmtList
               endWhile
               --> do
                   StmtList
               until Expr
               --> break
               --> continue
               --> return [ Expr ]
               --> for LValue = Expr to Expr [ by Expr ]
                   StmtList
               endFor
               --> for ( StmtList ; [ Expr ] ; StmtList )
                   StmtList
               endFor
               --> switch Expr
                   { case Expr : StmtList }
                   [ default : StmtList ]
               endSwitch
               --> try StmtList
                   { catch ID ParmList : StmtList }+
               endTry
               --> throw ID ArgList
               --> free Expr
               --> debug
```

# Context Free Grammar of KPL

```

Type          --> char
              --> int
              --> double
              --> bool
              --> void
              --> typeOfNull
              --> anyType
              --> ptr to Type
              --> record { Decl }+ endRecord
              --> array [ '[' Dimension { , Dimension } ']' ] of Type
              --> function ( [ Type { , Type } ] )
                  [ returns Type ]
              --> NamedType
NamedType     --> ID [ '[' Type { , Type } ']' ]
TypeList     --> NamedType { , NamedType }
Dimension    --> * | Expr
Constructor  --> Type ClassRecordInit
              --> Type ArrayInit
              --> Type
ClassRecordInit --> ID '{' ID = Expr { , ID = Expr } '}'
ArrayInit     --> ID '{' [ Expr of ] Expr
              { , [ Expr of ] Expr } '}'
LValue       --> Expr
Expr         --> Expr2 { ID : Expr2 }
Expr2       --> Expr3 { OPERATOR Expr3 }
Expr3       --> Expr5 { '|' Expr5 }
Expr5       --> Expr6 { '&&' Expr6 }
Expr6       --> Expr7 { '|' Expr7 }
Expr7       --> Expr8 { '^' Expr8 }
Expr8       --> Expr9 { '&' Expr9 }
Expr9       --> Expr10 { == Expr10
                  | != Expr10 }
Expr10      --> Expr11 { < Expr11
                  | <= Expr11
                  | > Expr11
                  | >= Expr11 }
Expr11      --> Expr12 { << Expr12
                  | >> Expr12
                  | >>> Expr12 }
Expr12      --> Expr13 { + Expr13
                  | - Expr13 }
Expr13      --> Expr15 { * Expr15
                  | / Expr15
                  | % Expr15 }
Expr15      --> OPERATOR Expr15
Expr16      --> Expr16
Expr17      --> Expr17 { . ID ArgList
                  . ID
                  | asPtrTo Type
                  | asInteger
                  | arraySize
                  | isInstanceOf Type
                  | isKindOf Type
                  | '[' Expr { , Expr } ']' }

```

# Context Free Grammar of KPL

```
Expr17      --> ( Expr )
            --> null
            --> true
            --> false
            --> self
            --> super
            --> INTEGER
            --> DOUBLE
            --> CHAR
            --> STRING
            --> NamelessFunction
            --> ID
            --> ID ArgList
            --> new Constructor
            --> alloc Constructor
            --> sizeof Type
```

## A simplified rule for expressions, which ignores precedence and associativity:

```
Expr        --> true
            --> false
            --> null
            --> self
            --> super
            --> INTEGER
            --> DOUBLE
            --> CHAR
            --> STRING
            --> ID ArgList
            --> ID
            --> NamelessFunction
            --> new Constructor
            --> alloc Constructor
            --> sizeof Type
            --> ( Expr )
            --> OPERATOR Expr
            --> Expr OPERATOR Expr
            --> Expr . ID ArgList
            --> Expr . ID
            --> Expr { ID : Expr }+
            --> Expr '[' Expr { , Expr } ']'
            --> Expr asPtrTo Type
            --> Expr asInteger
            --> Expr arraySize
            --> Expr isInstanceOf Type
            --> Expr isKindOf Type
```

# Context Free Grammar of KPL

```
===== (Lowest Precedence) =====
All keyword messages, e.g., x at:y put:z
=====
All infix operators not mentioned below
=====
||      Short-circuit for bool operands
=====
&&     Short-circuit for bool operands
=====
|      Bitwise OR for int operands
=====
^      Bitwise XOR for int operands
=====
&      Bitwise AND for int operands
=====
==     Can compare basic types, pointers, and
!=     objects, but not records or arrays
=====
<      Can compare int, double, and
<=     pointer operands
>
>=
=====
<<     Shift int operand left
>>     Shift int operand right arithmetic
>>>    Shift int operand right logical
=====
+      Can also add ptr+int
-      Can also subtract ptr-int and ptr-ptr
=====
*
/      For int, always truncates down, -7/3 => -3
%      Modulo operator for integers
=====
Prefix -      For int and double operands
Prefix !      For int and bool operands
Prefix *      Pointer dereference
Prefix &      Address-of
All other prefix methods
=====
.          Message Sending: x.foo(y,z)
.          Field Accessing: x.name
asPtrTo
asInteger
arraySize
isInstanceOf
isKindOf
[]         Array Accessing: a[i,j]
=====
()         Parenthesized expressions: x*(y+z)
constants e.g., 123, "hello"
keywords  i.e., true, false, null, self, super
nameless funs e.g., function(...)...endFunction
variables  e.g., x
function call e.g., foo(4)
new      e.g., new Person{name="smith"}
alloc    e.g., alloc Person{name="smith"}
sizeof   e.g., sizeof Person (in bytes)
===== (Highest Precedence) =====
```