A.2 Pragmas

#1 Pragma

```
pragma :
    "<*"
     ( encodingCtrlPragma[2] |
       conditionalPragma |
       compileTimeMessagePragma |
       codeGenerationPragma
       implementationDefinedPragma )
    "*>" ;
```

#2 Encoding Control Pragma Body

```
encodingCtrlPragma :
    ENCODING "=" encodingName[3] ( ":" codePointSampleList )? ;
encodingName : String ;
```

#3 CodePointSampleList

```
codePointSampleList :
    codePointSample ( "," codePointSample )* ;
```

#4 CodePointSample

```
codePointSample :
    quotedCharacterLiteral[4] "=" characterCodeLiteral ;
quotedCharacterLiteral : String ;
```

#5 Conditional Pragma Body

```
conditionalPragma :
    ( IF | ELSIF ) inPragmaExpression | ELSE | ENDIF ;
```

#6 Compile Time Message Pragma Body

```
compileTimeMessagePragma :
    INFO ( compileTimeMessage | ALIGN | implDefinedPragmaName ) |
    ( WARN | ERROR | FATAL ) compileTimeMessage ;
compileTimeMessage : String ;
```

#7 Code Generation Pragma Body

```
codeGenerationPragma :
    ALIGN "=" inPragmaExpression |
    FOREIGN ( "=" foreignInterfaceName[5] )? |
    MAKE "=" String |
    INLINE |
    NOINLINE |
    VOLATILE ;
foreignInterfaceName : String ;
```

#8 Implementation Defined Pragma Body

```
implementationDefinedPragma :
    implDefinedPragmaName ( "+" | "-" | "=" inPragmaExpression )? ;
implDefinedPragmaName : Ident[6] ;
```

---

[2] there may only be one encoding pragma per source file and it must occur before any other token.

[3] encoding identification strings are "ASCII" and "UTF8".

[4] a quoted character literal is a quoted string with one single character.

[5] the only foreign interface convention identification string defined at present is "C".

[6] implementation defined pragma names may only be lowercase or mixed case identifiers.

Copyright © 2010, 2011 B.Kowarsch & R.Sutcliffe. All rights reserved.

#9 In-Pragma Expression

```
inPragmaExpression :
    simpleInPragmaExpr ( inPragmaRelation simpleInPragmaExpr )? ;
```

#10 In-Pragma Relation

```
inPragmaRelation :
    "=" | "#" | "<" | "<=" | ">" | ">=" ;
```

#11 Simple In-Pragma Expression

```
simpleInPragmaExpr :
    ( "+" | "-" )? inPragmaTerm ( addOperator inPragmaTerm )* ;
```

#12 In-Pragma Term

```
inPragmaTerm :
    inPragmaFactor ( wholeNumberMulOperator inPragmaFactor )* ;
```

#13 Whole Number Multiply Operator

```
wholeNumberMulOperator :
    "*" | DIV | MOD | AND ;
```

#14 In-Pragma Factor

```
inPragmaFactor :
    wholeNumber[7] | constQualident[8] | "&" ( ALIGN | implDefinedPragmaName ) |
    "(" inPragmaExpr ")" | inPragmaPervasiveCall | NOT inPragmaFactor ;
wholeNumber : Number ;
```

#15 In-Pragma Pervasive Call

```
inPragmaPervasiveCall :
    ident[9] "(" inPragmaExpression ( "," inPragmaExpression ) ")" ;
```

---

[7] real number arithmetic is not supported within in-pragma expressions.

[8] only type identifiers, string constants, numeric constants of the Z-type and boolean constants are supported.

[9] callable pervasives and macros are ABS, NEG, ODD, ORD, EXP2, LENGTH, TSIZE, TMIN, TMAX, MIN and MAX.

Copyright © 2010, 2011 B.Kowarsch & R.Sutcliffe. All rights reserved.