*JFP* **13** (1): 145–146, January 2003. 
DOI: 10.1017/S095679680300131X Printed in the United Kingdom

# Chapter 11

# Compiler Pragmas

Some compiler implementations support compiler *pragmas*, which are used to give additional instructions or hints to the compiler, but which do not form part of the Haskell language proper and do not change a program's semantics. This chapter summarizes this existing practice. An implementation is not required to respect any pragma, but the pragma should be ignored if an implementation is not prepared to handle it. Lexically, pragmas appear as comments, except that the enclosing syntax is `{-# #-}`.

## 11.1 Inlining

$$
\begin{array}{lcl}
decl & \rightarrow & \texttt{\{-\#}\ \texttt{INLINE}\ qvars\ \texttt{\#-\}} \\
decl & \rightarrow & \texttt{\{-\#}\ \texttt{NOINLINE}\ qvars\ \texttt{\#-\}}
\end{array}
$$

The `INLINE` pragma instructs the compiler to inline the specified variables at their use sites. Compilers will often automatically inline simple expressions. This may be prevented by the `NOINLINE` pragma.

## 11.2 Specialization

$$
\begin{array}{lcll}
decl & \rightarrow & \texttt{\{-\#}\ \texttt{SPECIALIZE}\ spec_1\ \texttt{,}\ \ldots\ \texttt{,}\ spec_k\ \texttt{\#-\}} & (k \geq 1) \\
spec & \rightarrow & vars\ \texttt{::}\ type &
\end{array}
$$

Specialization is used to avoid inefficiencies involved in dispatching overloaded functions. For example, in

```
factorial :: Num a => a -> a
factorial 0 = 0
factorial n = n * factorial (n-1)
{-# SPECIALIZE factorial :: Int -> Int,
               factorial :: Integer -> Integer #-}
```

calls to `factorial` in which the compiler can detect that the parameter is either `Int` or `Integer` will use specialized versions of `factorial` which do not involve overloaded numeric operations.

https://doi.org/10.1017/S095679680300131X Published online by Cambridge University Press