If the conditional expression is not contextually typed, the result is of the best common type of the types of *Expr1* and *Expr2*. An error occurs if the best common type is not identical to at least one of the two candidate types.

4.17 Assignment Operators

An assignment of the form

VarExpr = *ValueExpr*

requires *VarExpr* to be classified as a reference (section 4.1). *ValueExpr* is contextually typed (section 4.19) by the type of *VarExpr*, and the type of *ValueExpr* must be assignable to (section 3.8.4) the type of *VarExpr*, or otherwise a compile-time error occurs. The result is a value with the type of *ValueExpr*.

A compound assignment of the form

VarExpr Operator= *ValueExpr*

where *Operator*= is of the compound assignment operators

```
*=    /=    %=    +=    -=    <<=    >>=    >>>=    &=    ^=    |=
```

is subject to the same requirements, and produces a value of the same type, as the corresponding non-compound operation. A compound assignment furthermore requires *VarExpr* to be classified as a reference (section 4.1) and the type of the non-compound operation to be assignable to the type of *VarExpr*.

4.18 The Comma Operator

The comma operator permits the operands to be of any type and produces a result that is of the same type as the second operand.

4.19 Contextually Typed Expressions

In certain situations, parameter and return types of function expressions are automatically inferred from the contexts in which the function expressions occur. For example, given the declaration

```
var f: (s: string) => string;
```

the assignment

```
f = function(s) { return s.toLowerCase(); }
```

infers the type of the 's' parameter to be the String primitive type even though there is no type annotation to that effect. The function expression is said to be ***contextually typed*** by the variable to which it is being assigned. Contextual typing occurs in the following situations: