17 Templates [temp]

1 A *template* defines a family of classes, functions, or variables, an alias for a family of types, or a concept.

template-declaration:
template-head declaration
template-head concept-definition

template-head:
`template <` *template-parameter-list* `>` *requires-clause*$_{opt}$

template-parameter-list:
template-parameter
template-parameter-list `,` *template-parameter*

requires-clause:
`requires` *constraint-logical-or-expression*

constraint-logical-or-expression:
constraint-logical-and-expression
constraint-logical-or-expression `||` *constraint-logical-and-expression*

constraint-logical-and-expression:
primary-expression
constraint-logical-and-expression `&&` *primary-expression*

concept-definition:
`concept` *concept-name* `=` *constraint-expression* `;`

concept-name:
identifier

[*Note:* The `>` token following the *template-parameter-list* of a *template-declaration* may be the product of replacing a `>>` token by two consecutive `>` tokens (17.2). — *end note*]

2 The *declaration* in a *template-declaration* (if any) shall

(2.1) — declare or define a function, a class, or a variable, or

(2.2) — define a member function, a member class, a member enumeration, or a static data member of a class template or of a class nested within a class template, or

(2.3) — define a member template of a class or class template, or

(2.4) — be a *deduction-guide*, or

(2.5) — be an *alias-declaration*.

3 A *template-declaration* is a *declaration*. A *template-declaration* is also a definition if its *template-head* is followed by either a *concept-definition* or a *declaration* that defines a function, a class, a variable, or a static data member. A declaration introduced by a template declaration of a variable is a *variable template*. A variable template at class scope is a *static data member template*.

[*Example:*

```
template<class T>
  constexpr T pi = T(3.1415926535897932385L);
template<class T>
  T circular_area(T r) {
    return pi<T> * r * r;
  }
struct matrix_constants {
  template<class T>
    using pauli = hermitian_matrix<T, 2>;
  template<class T>
    constexpr pauli<T> sigma1 = { { 0, 1 }, { 1, 0 } };
  template<class T>
    constexpr pauli<T> sigma2 = { { 0, -1i }, { 1i, 0 } };
```