A UDL generally consists of two parts: (1) a valid lexical literal token and (2) a user-defined suffix. The signature of each UDL operator must conform to one of three patterns, distinguished by the way the compiler supplies the naked literal to the UDL operator.

1. **Cooked UDL operator** — The naked literal is evaluated at compile time and passed into the operator as a value:

   ```
   Type1 operator""_t1(unsigned long long n);
   Type1 t1 = 780_t1;  // calls operator""_t1(780ULL)
   ```

2. **Raw UDL operator** — The characters that make up the naked literal are passed to the operator as a *raw*, unevaluated string (for numeric literals only):

   ```
   Type2 operator""_t2(const char* token);
   Type2 t2 = 780_t2;  // calls operator""_t2("780")
   ```

3. **UDL operator template** — The UDL operator is a template whose parameter list is a variadic sequence of **char** values (see Section 2.1."Variadic Templates" on page 873) that make up the naked literal (for numeric literals only):

   ```
   template <char...> Type3 operator""_t3();
   Type3 t3 = 780_t3;  // calls operator""_t3<'7', '8', '0'>()
   ```

Each of these three forms of UDL operators is expounded in more detail in its own separate section; see *Cooked UDL operators* on page 843, *Raw UDL operators* on page 845, and *UDL operator templates* on page 849.

When a UDL is encountered, the compiler prioritizes a cooked UDL operator over the other two. Given a UDL having suffix `_udl`, the compiler will look for any **`operator`**`""_udl` in the **local scope** (**unqualified name lookup**). If, among the operators found, there is a cooked UDL operator that exactly matches the type of the naked literal, then that UDL operator is called. Otherwise, for numeric literals only, the raw UDL operator or a UDL operator template is invoked; an ambiguity results if both operators are found. This set of lookup rules is deliberately short and rigid. Importantly, this lookup sequence differs from other operator invocations in that it does *not* involve overload resolution or argument conversions, nor does it employ **argument-dependent lookup (ADL)** to find operators in other namespaces.

Although ADL is never an issue for UDLs, common practice is to gather related UDL operators into a namespace (whose name often contains the word "literals"). This namespace is then typically nested within the namespace containing the definitions of the user-defined types that the UDL operators return. These literals-only nested namespaces enable a user to import just the literals into their scope via a single **`using`** directive, thereby substantially decreasing the likelihood of collisions with names in the enclosing **`namespace`**: