With that said, using **auto** to deduce references to built-in arrays is straightforward:

```
int data[] = {1, 2};

      auto&  arr6 = data;                          //       int (&) [2]
const auto&  arr7 = BuiltInArray<int, 2>{1, 2};  // const int (&) [2]
      auto&& arr8 = BuiltInArray<int, 2>{1, 2};  //       int (&&)[2]
```

Note that the `arr7` and `arr8` references in the code snippet immediately above extend the lifetime of the temporary arrays that they bind to, so subscripting them does not have the undefined behavior that subscripting `arr5` in the previous code snippet has.

Annoyances

auto is disallowed for nonstatic data members

Despite C++11 allowing nonstatic data members to be initialized within class **definitions**, **auto** cannot be used to declare them:

```
class C
{
    auto d_i = 1;  // Error, nonstatic data member is declared with auto.
};
```

Not all template argument deduction constructs are allowed for auto

Despite **auto** type deduction largely following the template argument deduction rules, certain constructs that are allowed for templates are not allowed for **auto**. For example, when deducing a pointer-to-data-member type, templates allow for deducing both the data member type and the **class type**, whereas **auto** can deduce only the former:

```
struct Node
{
    int   d_data;
    Node* d_next;
};

template <typename TYPE>
void deduceMemberTypeFn(TYPE Node::*);

void testDeduceMemberType()
{
                deduceMemberTypeFn   (&Node::d_data);  // OK, int Node::*
    auto Node::* deduceMemberTypeVar = &Node::d_data;   // OK,  "     "
}

template <typename TYPE>
void deduceClassTypeFn(int TYPE::*);
```