Unlike references, pointer types can be deduced by **auto** alone. Therefore, different forms of **auto** can be used to declare a variable of a pointer type:

```
auto  cptr1 = &cval;  // const int*
auto* cptr2 = &cval;  //   "     "

auto   fptr1         = &freeF;  // float (*)(float)
auto  *fptr2         = &freeF;  //   "         "
auto (*fptr3)(float) = &freeF;  //   "         "

auto      mptr1 = &S::d_data;  // double S::*
auto S::* mptr2 = &S::d_data;  //    "     "

auto       mfptr1        = &S::memberF;  // int (S::*)(long)
auto (S::* mfptr2)(long) = &S::memberF;  //  "     "     "
```

Note, however, that because regular and member pointers are fundamentally different in the C++ type system, **auto*** cannot be used to deduce pointers to data members and member functions:

```
auto* mptr3  = &S::d_data;   // Error, cannot deduce auto* from &S::d_data
auto* mfptr3 = &S::memberF;  // Error, cannot deduce auto* from &S::memberF
```

Pointers might also be deduced from array and function initializers without explicit use of the address-of operator due to function-to-pointer and array-to-pointer conversions applied prior to deduction of nonreference types:

```
auto   fptr4         = freeF;  // float (*)(float)
auto  *fptr5         = freeF;  //   "         "
auto (*fptr6)(float) = freeF;  //   "         "

int array[4];

auto  aptr1 = array;  // int*
auto* aptr2 = array;  //   "

auto  sptr1 = "hello";  // const char*
auto* sptr2 = "world";  // const char*
```

These conversions are not applied when deducing a reference type, and function and array references are deduced instead:

```
auto& fref = freeF;   // float (&)(float)

auto& aref = array;   // int (&)[4]

auto& sref = "meow";  // const char (&)[5]
```

Storage class specifiers as well as the **constexpr** (see Section 2.1."**constexpr** Variables" on page 302) specifier can also be applied to variables that use **auto** in their declaration: