```
auto&& rref = doStuff();
    // Type of rref is deduced to be double&&.
```

Similarly to references, explicitly specifying that a pointer type is to be deduced is possible. If the supplied initializer is not a pointer type, the compiler will issue an error:

```
const auto* cptr = &val;
    // Type of cptr is deduced to be const int*,
    // the same as the argument for template <typename T> void deducer(const T*).

auto* cptr2 = cval;  // Error, cannot deduce auto* from cval
```

The compiler can also be instructed to deduce pointers to functions, data members, and member functions, but see *Annoyances — Not all template argument deduction constructs are allowed for* ***auto*** on page 212:

```
float freeF(float);

struct S
{
    double d_data;
    int memberF(long);
};

auto (*fptr)(float) = &freeF;
  // Type of fptr is deduced to be float (*)(float),
  // the same as the argument for template <typename T> void deducer(T (*)(float)).

const auto S::* mptr = &S::d_data;
  // Type of mptr is deduced to be const double S::*,
  // the same as the argument for template <typename T> void deducer(const T S::*).

auto (S::* mfptr)(long) = &S::memberF;
  // Type of mfptr is deduced to be int (S::*)(long),
  // the same as the argument for template <typename T> void deducer(T (S::*)(long)).

auto (*gptr)(float) = 2; // Error, must be a function address

float freeH(double) { return 0.0; }

auto (*hptr)(float) = &freeH; // Error, the function must have the
                              // specified parameters.

double freeG(float-) { return 0.0; }

auto (*itpr)(float) = &freeG;  // OK, the return value is not constrained.
```