```
decltype(auto) g10(int i)
{
    if (i > 1) { return i + g10(i - 1); }  // Error, return type not known yet
    else       { return 0; }
}
```

Perhaps surprisingly, `g9` cannot be rewritten using the ternary conditional operator because return-type deduction cannot occur until both the **`true`** and **`false`** branches of the ternary expression have been processed by the compiler:

```
decltype(auto) g11(int i)  // erroneous rewrite of g9
{
    return i < 1 ? 0 : i + g11(i - 1);
        // Error, g11 used before return deduced
}
```

It is not necessary to provide a **`return`** statement at the end of the function if non**`void`** return type has already been deduced. However, the control flow falling off the end of the function has **undefined behavior**:

```
auto g12(bool b) { if (b) return 1;         }  // Bug, UB if b is false
auto g13(bool b) { if (b) return 1; return; }  // Error, deduction mismatch
```

Type of a function having a deduced return type

Deduced return types are allowed for almost every category of function, including free functions, static member functions, nonstatic member functions, function templates, member function templates, and conversion operators. Virtual functions, however, cannot have deduced return types:

```
auto free();                                    // OK, free function
template <typename T> auto templ();             // OK, function template

struct S
{
    static auto staticMember();                 // OK, static member function
    decltype(auto) member();                    // OK, nonstatic member function
    template <typename T> auto memberTempl();   // OK, member function template
    operator auto() const;                      // OK, conversion operator
    virtual auto virtMember();                  // Error, virtual function
};
```

When one of these functions is later defined or redeclared, it must use the *same* placeholder for the return type, even if the actual return type is known at the point of definition: