【C++】Day seven(2)

Class	C++
Date	@November 25, 2021
Material	
Series Number	
Summary	

【Ch2】Decltype Specifier

2.5.3 The decltype Type Specifier

Sometimes we want to define a variable with a type that the compiler deduces from an expression but do not want to use that expression to initialize the variable.

In such cases, the new standard introduced a second type specifier, `decltpe`, which returns the type of its operand. The compiler analyses the expression to determine its type but does not evaluate the expression:

```
decltype(f()) sum = x; //sum has whatever type f returns
```

The compiler does not call f(), but it uses the type that such a call would return as the type for sum. That is, the compiler gives sum the same type as the type that would be returned if we were to call f().

The way `decltype` handles top-level `const` and references differs subtly from the way auto does.

When the expression to which we apply `decltype` is a variable, `decltype` returns the type of that variable, including top-level `const` and references:

```
const int ci = 0, &cj = ci;
decltype(ci) x = 0; //x has type const int
```

```
decltype(cj) y = x; //y has type const int& and is bound to x
decltype(cj) z; //error: z is a reference and must be initialized.
```

Note: `decltype` is the only context in which a variable defined as a reference is not treated as a synonym for the object to which it refers.

Generally speaking, `decltype` returns a reference type for expressions that yield objects that can stand on the left-hand side of the assignment:

```
//decltype of an expression can be a reference type
int i = 42, *p = &i, &r = i;
decltype(r + 0) b; //ok: addition yields an int; b is an uninitialized int
decltype(*p) c; //error: c is int& and must be initialized
```

Here r is a reference, so `decltype(r)` is a reference type. If we want the type to which r refers, we can use r in an expression, such as `r + 0`, which is an expression that yields a value that has a nonreference type.

On the other hand, the dereference operator `*` is an example of an expression for which `decltype` returns a reference. As we've seen, when we dereference a pointer, we get the object to which the pointer points. Moreover, we can assign to that object. Thus, the type deduced by `decltype (*p)` is `int&`, not plain `int`.

Another important difference between `decltype` and `auto` is that the deduction done by `decltype` depends on the form of its given expression. When we apply `decltype` to a variable without any parentheses, we get the type of that variable. If we wrap the variable's name in one or more sets of parentheses, the compiler will evaluate the operand as an expression. As a result, `decltype` on such an expression yields a reference;

```
decltype ((i)) d; //error: d is int& and must be initialized
decltype(i) b; //ok: b is an uninitialized int
```

Assignment is an example of an expression that yields a reference type. The type is a reference to the type of the left-hand operand. That is, if `i` is an `int`, then the type of the expression `i = x` is `int&`.