Why are the C and C++ compilers giving me error messages about int when my code doesn't mention int?

devblogs.microsoft.com/oldnewthing/20201230-00

December 30, 2020

Raymond Chen

You're trying to get your code to compile without errors, and you're working through the error list, and then you get to some error message that complains about `int` when your code never mentions `int`:

```
void f()
{
  const char* p = get_user_name();
}
```

The errors are

```
test.cpp(3): error C2065: 'get_user_name' : undeclared identifier
```

Okay, I expected that one. Forgot to declare `const char* get_user_name();`. I know how to fix that: Add the declaration.

Now to the next error:

```
test.cpp(3) : error C2440: 'initializing' : cannot convert from 'int' to 'const char
*'
  Conversion from integral type to pointer type requires reinterpret_cast, C-style
cast or function-style cast
```

What's this about converting an `int` to a `const char *`? There are no `int`s in this code anywhere!

What you're seeing is a cascade error. The compiler didn't see any declaration for `get_user_name()`, so it had two choices.

1. Stop the compilation right there.
2. Keep going to see if there are any more errors.

Nearly all compilers go for the second option, because duh. But this means that the compiler needs to recover from its error state into some sort of valid state.

A popular choice for recovery is to assume that all undeclared variables are of type `int`, and all undeclared functions return `int`. This is probably for historical reasons, because in the original C language, there were a lot of places where if you didn't specify a type, you got `int`. In particular, you could call a function without declaring it, and the function was assumed to return `int`.

In those cases, the compiler recovers by inserting the declaration

```
extern int get_user_name();
```

and resuming. And that leads to the second error message: You're trying to assign an `int` to a `const char *`.

Of course, that `int` is not your `int`. It's the `int` that the compiler's error recovery machinery manufactured out of nowhere in a futile attempt to get the compiler back on track.

Sometimes, the compiler is nice enough to tell you what its error recovery is doing, with a message like "undeclared identifier, assuming int." But sometimes it just plows forward without telling you what recovery steps it took.

At some point[1] between Visual Studio 6 and 2017,[2] the Microsoft compiler changed the way it recovers from undefined identifiers. Instead of assuming that they are `int`, it treats them as a hypothetical type called `unknown-type`.

So if you see compiler errors about `unknown-type`, then it's the same problem: The compiler encountered an earlier error and created some imaginary declarations to try to get itself back on track so it can report additional errors.

I hope at least that the name `unknown-type` makes it clearer what happened.

[1] I was too lazy to install every version of Visual Studio between 6 and 2017. You can get your refund at the counter over there.

[2] Depending on how you count, this means that it's a range of 9 versions or 2011 versions.

Raymond Chen

Follow