WG14 N3454

Author: José Miguel Sánchez García (soy.jmi2k@gmail.com)

Date: 2025-01-21

Project: ISO/IEC JTC1/SC22/WG14 9899: Programming Language — C

Proposal category: Change Request, Feature Request

Target: General Developers, Compiler Developers

`typeof(return)`

Summary

Change the definition of the `typeof` operator, allowing the `return` keyword to be used when the operator is used inside the parameter list of a function declaration/definition, or inside the body of a function definition, to obtain its return type; and emitting an error message if used in any other situation.

Rationale

C23 introduces both the `auto` keyword and the `typeof` operator. These can be combined to facilitate using structs or unions without a tag as the return type. This has the potential to improve the ergonomics of multiple return values in situations where both output parameters and named struct/unions are not a good fit.

```
struct {
    int x;
    char c;
}
auxiliary_function(int n)
{
    return (typeof(auxiliary_function(n))){ n / 5, 'V' };
}

int
main(int argc, char **argv)
{
    auto result = auxiliary_function(42);
    return result.x == result.c;
}
```

This approach is superior to both alternatives (which are currently common practice):

1. Output parameters can be undesirable due to the need to rely on pointers to memory, or even not permitted due to ABI requirements for that particular function.
2. Structs and unions with a tag pollute the namespace with definitions that may not be useful beyond that particular caller/callee boundary. While it is technically feasible to use named structs/unions in this case, it remains an annoyance where a dummy definition with a meaningless name must be chosen to appease the language.

However, it faces one final hurdle: getting the return value of a function inside the function itself remains too cumbersome. In particular, the fact that the entire function name and arguments have to be typed is enough to deter anyone from choosing this approach.

Also, according to the current specification, it remains impossible to refer to such a return type from the parameter list of the function declaration (as it is not yet declared, `typeof` cannot refer to it).

While an extra type identifier could be introduced to refer to such type, any such suggestion risks conflicting with existing code. Hence, a safer approach is proposed: extending the definition of the existing `typeof` operator, allowing the `return` keyword to refer to the type of the enclosing function. This causes no conflict outside said operator (as no new keyword is introduced) nor inside it (as `return` is currently not allowed in that position), and is immediately recognizable for what it represents (the type of the return value). Also, this choice of reusing existing keywords in a different context aligns with other existing features, like those repurposing `static` and `*` in array parameter declarations.

Proposed changes

This shows proposed **additions** and ~~removals~~ relative to WG14 N3435:

- (6.7.3.6, #1) add the new `return` keyword to the *typeof-specifier-argument* definition:

  typeof-specifier-argument:
  expression
  type-name
  return

- (6.7.3.6, #3) add an additional constraint on the usage of `typeof(return)`:

  The typeof operators shall not be applied to an expression that designates a bit-field member. **The return keyword shall only be used inside the typeof operator if that operator appears inside the parameter list of a function declaration, inside the parameter list of a function definition, or inside the body of a function definition.**

- (6.7.3.6, #4) extend the semantics of the `typeof` to accommodate this proposal:

  The typeof specifier applies the typeof operators to an *expression* (6.5.1) ~~or~~, a type name **or the return keyword**. If the typeof operators are applied to an expression, they yield the type of their operand.[148)] **If the typeof operators are applied to the return keyword inside a function parameter list or inside a function body, they yield the return type of that function.** Otherwise, they designate the same type as the type name with any nested typeof specifier evaluated.[149)] If the type of the operand is a variably modified type, the operand is evaluated; otherwise, the operand is not evaluated.

- (6.7.3.6) add an example that showcases the added functionality (paragraph #13):

EXAMPLE 8 Return type of the current function.

```
int main(typeof(return), char **);

int main(typeof(return) argc, char **argv) {
      typeof(return) result = argc & 0xF;
      return result;
}
```

is equivalent to this program:

```
int main(int, char **);

int main(int argc, char **argv) {
      int result = argc & 0xF;
      return result;
}
```