```
#include "clinic/_pickle.c.h"
```

13. Double-check that the argument-parsing code Argument Clinic generated looks basically the same as the existing code.

    First, ensure both places use the same argument-parsing function. The existing code must call either `PyArg_ParseTuple()` or `PyArg_ParseTupleAndKeywords()`; ensure that the code generated by Argument Clinic calls the *exact* same function.

    Second, the format string passed in to `PyArg_ParseTuple()` or `PyArg_ParseTupleAndKeywords()` should be *exactly* the same as the hand-written one in the existing function, up to the colon or semi-colon.

    (Argument Clinic always generates its format strings with a `:` followed by the name of the function. If the existing code's format string ends with `;`, to provide usage help, this change is harmless—don't worry about it.)

    Third, for parameters whose format units require two arguments (like a length variable, or an encoding string, or a pointer to a conversion function), ensure that the second argument is *exactly* the same between the two invocations.

    Fourth, inside the output portion of the block you'll find a preprocessor macro defining the appropriate static `PyMethodDef` structure for this builtin:

```
#define __PICKLE_PICKLER_DUMP_METHODDEF    \
{"dump", (PyCFunction)__pickle_Pickler_dump, METH_O, __pickle_Pickler_dump__
↪doc__},
```

    This static structure should be *exactly* the same as the existing static `PyMethodDef` structure for this builtin.

    If any of these items differ in *any way*, adjust your Argument Clinic function specification and rerun `Tools/clinic/clinic.py` until they *are* the same.

14. Notice that the last line of its output is the declaration of your "impl" function. This is where the builtin's implementation goes. Delete the existing prototype of the function you're modifying, but leave the opening curly brace. Now delete its argument parsing code and the declarations of all the variables it dumps the arguments into. Notice how the Python arguments are now arguments to this impl function; if the implementation used different names for these variables, fix it.

    Let's reiterate, just because it's kind of weird. Your code should now look like this:

```
static return_type
your_function_impl(...)
/*[clinic end generated code: checksum=...]*/
{
...
```

    Argument Clinic generated the checksum line and the function prototype just above it. You should write the opening (and closing) curly braces for the function, and the implementation inside.

    Sample:

```
/*[clinic input]
module _pickle
class _pickle.Pickler "PicklerObject *" "&Pickler_Type"
[clinic start generated code]*/
/*[clinic end generated code:␣
↪checksum=da39a3ee5e6b4b0d3255bfef95601890afd80709]*/

/*[clinic input]
_pickle.Pickler.dump

    obj: 'O'
        The object to be pickled.
```

(continues on next page)