Recursive macro expansion in compilers refers to the process by which macros, which are essentially shorthand notations for longer code sequences, are expanded recursively until no more macro invocations remain. This process is a fundamental aspect of the macro preprocessing stage in compilers, particularly in languages like C and C++.

Here's how recursive macro expansion typically works in compilers:

1. **Initial Macro Invocation**: When a macro is invoked in the source code, the compiler identifies it during the preprocessing stage. The macro invocation is replaced with the corresponding macro definition.
2. **Macro Expansion**: If the macro definition contains other macro invocations, they are also expanded. This process continues recursively until no more macro invocations remain in the expanded code.
3. **Stopping Condition**: The recursive expansion stops when there are no more macros to expand or when a macro definition is self-referential (i.e., when a macro invokes itself), which would lead to infinite recursion.
4. **Handling Arguments**: Macros can also accept arguments, and these arguments can be expressions or other macro invocations. Recursive expansion takes place within macro arguments as well.
5. **Resultant Code**: After all macro expansions are complete, the resulting code contains the expanded versions of all macro invocations. This expanded code is then passed on to subsequent stages of compilation.

It's worth noting that while recursive macro expansion can be powerful, it can also lead to code bloat and potential pitfalls, especially if not used judiciously. Therefore, it's essential to understand the implications of macro expansion and use it wisely in software development.

```
#define SQUARE(n) ((n) * (n))

#define FOUR_TIMES(x) (4 * (x))

int main() {
    int a = MAX(5, 8);            // Expands to: ((5) > (8) ? (5) : (8))
    int b = SQUARE(4 + 1);        // Expands to: ((4 + 1) * (4 + 1))
    int c = FOUR_TIMES(3 + 2);    // Expands to: (4 * (3 + 2))
    return 0;
}
```

In this example, the macros **`MAX`**, **`SQUARE`**, and **`FOUR_TIMES`** are recursively expanded into their respective definitions during the preprocessing stage, resulting in the expanded code that will be passed to the subsequent stages of compilation.

then explain which I already send you and also from the book