Here are some hints for the interpretor part of implementing "break" and "continue". When the interpretor encounters a CONTINUE, it could set a flag which will indicate to all subsequent ex()'s to skip their execution, until the flag is reset by a WHILE, FOR, or DO-WHILE loop at the end of their body. Similarly for the BREAK. This is another good example of why interpretation is slower than compiled code: in the compiler version, the generated code would simply "jump" over the remaining lines instead of having to check and "skip" each one of them in succession.

I have the following declarations in my "c5i.c". I use some aux. variables to indicate whether a break or continue has been encountered during the course of executing a while, for, ... or switch. They will then affect the execution of the remaining subtrees inside the loop/switch.

```
static int cont = 0; // = 1 if "continue" has been encountered
static int brk = 0; // = 1 if "break" has been encountered
static int scope = 0; // outermost scope - no continue/break allowed
int ex(nodeType *p) { ...
```

As for the compiler, I added two labels as parameters to be passed down to subtrees in my "c5c.c"; they represent the entry location (for "continue" to jump to) and the exit location (for "break" to jump to) of a loop (or switch) structure.

```
// Added label l1 for continue to jump to, l2 for break to jump to,
// if applicable (else -1, -1 are assigned):
int ex(nodeType *p, int l1, int l2) { ...
```