Interpreter

<<Interpreter.ppt>>

Interpreters allow the following

- Modification of or addition to user programs as execution proceeds. This facility provides a straightforward interactive debugging capability. Such modification is easiest in non-block-structured languages such as APL or BASIC, because individual statements can be changed without reparsing an entire program.

Interpreters allow the following (cont.)

- Languages in which the type of object that a variable denotes may change dynamically. The user program is continuously reexamined as execution proceeds, and symbols need not have a fixed meaning (that is, a symbol may denote an integer scalar at one point and a boolean array at a later point). Such fluid bindings are obviously much more troublesome for compilers, as dynamic changes in the meaning of a symbol make direct translation into machine code impossible.

Interpreters allow the following (cont.)

- Better diagnostics. Because source text analysis (normally done at compile-time) is intermixed with execution of the program, especially good diagnostics (recreation of source lines in error, use of variable names in error messages, and so on) are produced more easily than they are by compilers.

Interpreters allow the following (cont.)

- A significant degree of machine independence, since no machine code is generated. All operations are performed within the interpreter. Therefore, to move an interpreter, we need only recompile it on a new machine.

Interpretation can involve large overheads

- As execution proceeds, program text must be continuously reexamined, with identifier bindings, types, and operations potentially being reconsidered at each reference. For very dynamic languages this can represent a 100:1 (or worse) factor in execution speed. For more static languages (such as BASIC), the speed degradation is closer to 10:1.

Interpretation can involve large overheads (cont.)

- Substantial space overhead may be involved. The interpreter and all support routines must usually be kept available. This program representation is often not as compact as compiled machine code (for example, symbol tables are present, and program text may be stored in a format designed for easy access and modification rather than for space minimization). This size penalty may lead to restrictions in the size of programs, the number of variables or procedures, and so on. Programs beyond these built-in limits cannot be handled by the interpreter.

Interpretation can involve large overheads (cont.)

- Processes compiler control directives (turn the listing on or off, include source images from a file, and so on). These directives are often done via pseudocomments that have the syntactic form of a comment but include special information intended for processing by the compiler. Another approach is that used in Ada, where a special syntactic structure, the pragma, is used for this purpose.

Interpretation can involve large overheads (cont.)

- Enters preliminary information into symbol and attribute tables (for example, to produce a later cross-reference listing).
- Formats and lists the source program.