Pascal/66 compiler. That is, for the compilation of a given level of the graphics package, the compiler must be supplied with the types of procedures called from other levels and which will eventually be linked to the library created for that level. The informal distinction between private and exported procedures now becomes essential. If so directed, Pinc can traverse additional trees and extract exactly the headers of the procedures appearing below an EXPORT catalog. This guarantees that the procedure declarations used by the calling code are identical to that used in the declaration of the called procedure and that only those procedures which are designated exportable can indeed be accessed externally. This approach is not 100% secure in the event of rapidly changing declarations. But given that type-checking is available only at compile-time and not at link-time, it is far superior to attempting to keep declarations consistent between levels by ad hoc means.

A more frustrating problem is presented by the need for *forward* declarations. In Pascal, all procedures must be declared before use and this results in the need for forward declarations, in which the type declaration and the remainder of a procedure's declaration are separated to admit mutual recursion. Ideally, one would like Pinc to analyze the procedure dependencies, and determine how to declare or order all the procedures within a level of the package, without programmer intervention. However, this level of sophistication is not yet our goal and Pinc at present only handles the ordering of procedure declarations when the programmer divides the HEADER file into a HEADER and a FORWARD file containing a forward declaration. All forward declarations found within a level are placed in the inclusion file to precede the definitions of all the procedure bodies themselves. Though it works tolerably well, we feel a better solution is needed. This illustrates very poignantly the problems of writing production software in Pascal: a sophisticated tool is required to present naturally organized source text to a one-pass compiler.

Discussion

Experience to date with the use of Pinc on the graphics package is limited. However, it is already clear its advantages go beyond the expected savings of programmer time and machine resources that would otherwise be expended with multiple versions of the package. Two major benefits emerge immediately.

First, programmers working on the package perceive a much clearer view of the the nature of the package, both its global structure and its detailed variation. Though no empirical results have been sought, we are convinced that this clearer understanding leads to more accurate and reliable maintenance of the the software. Comprehension of complex source text is a major obstacle faced by the maintenance programmer, and this is compounded by the presence of multiple versions of the software. Improving the presentation of the source to the programmer and reducing the presence of duplicate source code alleviates some of this burden and makes it easier to concentrate on the logic.

Second, the need to decide formally on how to structure the package forces decisions, each of which better clarifies the source. Versions must be characterized and named meaningfully, and as with identifiers within programs the right choice for a name can be critical. Procedures must be designated as private or exportable. Such decisions force the programmer to think consciously about aspects of the software which may be blurred when ad hoc solutions are employed. What this amounts to is the introduction of programming style in-the-large. An elementary configuration language is available and the issues of stylistic expression must be resolved. As with style issues in-the-small, time spent during program synthesis pays dividends for the reader later on.

Furthermore, this organization is easily comprehended when first introduced to programmers, since it follows the structure of both Pascal and the graphics package. This results in it being adopted enthusiastically as its benefits are perceived.

Conclusion

Through the use of a relatively simple organization for storing Pascal source text and a simple tool for selecting subsets of this source for compilation, the management of a large and evolving computer graphics package has been enhanced. Based on a general purpose file system in a conventional computing environment, the proposed technique is both inexpensive and easily implemented.

Acknowledgement

We wish to thank J.C. Beatty and K.S Booth, the original authors of the graphics package, for furnishing the motivation for this work, a vehicle with which to conduct it and critical reviews of our progress.

References

1 Beatty J.C., Booth K.S. "Teaching Computer Graphics at the University Waterloo", Proc. American Society for Engineering Education Annual Conf. 1981.

2 Buxton J.N. "Stoneman: Requirements for Ada Programming Support Environments", US Department of Defense, 1980.

3 Cargill T.A. "Management of the Source Text of a Portable Operating System", Proc. 4th Int. Computer Software and Applications Conf. pp 674-768 IEEE, 1980.

4 Cargill T.A. "A View of Source Text for Diversely Configurable Software", Ph.D. Thesis, Technical Report CS-79-28, Department of Computer Science, University of Waterloo 1979.

5 DeRemer F., Kron H. "Programming in-the-large versus programming in-the-small" Proc. Int. Conf. on Reliable Software, SIGPLAN Notices 10:6 pp 114-121 1975.

6 Gustafson G.C. et al. "Some Practical Experience with the Pascal Language", Proc. National Computer Conference Vol 48, AFIPS, 1980.

7 Hall D.E. et al. "A Virtual Operating System", Communications of the ACM 23:9 pp 495-502 1980.

8 Jensen K., Wirth N. *Pascal User Manual and Report* Springer-Verlag 1974.

9 Wasserman A.I. (editor) *Programming Environments* Computer 14:4 1980.