

## Abstract

The Database Program Generator (DPG) is a system for creating database software. It is designed to be used by professional programmers and analysts, as opposed to end users; however, it is designed to make the process of constructing and maintaining database software much less time-consuming than it would otherwise be.

Programs created using the system are first written in a fourth-generation language (4GL) which is specifically designed for the purpose. They are then translated into the C language, which is a general purpose programming language available on almost all computer systems. The C code is compiled and linked with some run-time support code to create a stand-alone program which can be run by an end user without access to DPG itself.

DPG is able to create programs which manipulate data consisting of fixed-length records containing textual and numeric information. It is not designed to create programs which handle such data types as graphics, sound, large text files, hypertext etc. However, it is possible to write code to handle these data types in C and link this code in with a DPG application.

The motivation for writing the system is to fill the gap between database management systems (DBMSs) which provide data handling functions, but require the rest of an application program to be written in a conventional programming language, and fourth-generation language packages (4GLs) such as dBase, which are designed to be used by people who are not professional programmers, and which therefore have seriously limited functionality. DPG provides a procedural language with power equivalent to that of a conventional programming language, but provides support for the development of complete application programs, not only the data handling functions.

## Contents

μAbstract	1
Contents	2
1 Introduction	3
2 Background	3
2.1 Data Models	3
2.1.1 The Hierarchical Model	3
2.1.2 The Network Model	4
2.1.3 The Relational Model	6
2.2 Database Management Systems	7
2.3 Fourth Generation Languages	7
2.4 A Procedural Application Generator	7
3 Design and Implementation	8
3.1 File Structure	9
3.2 Application Development	11
3.3 Portability	12
4 Conclusions	12
5 Future Work	13
Bibliography	14
Appendix A DPG Language Grammar	15
A.1 Modified BNF	15
A.2 DPG Language	15
Appendix B DPG Language User Manual	18
B.1 Introduction	18
B.1.1 Keywords	18
B.1.2 Identifiers	18
B.1.3 Constants	18
B.1.4 Strings	19
B.1.5 Other Symbols	19
B.2 Program Composition	19
B.3 Variable Declarations	19
B.4 File Definitions	20
B.4.1 Fields	20
B.4.2 Index	20
B.4.3 Linked On	20
B.4.4 Table	21
B.5 Procedures	21
B.6 Statements	21
B.7 Directions	25
B.8 Expressions	26
Appendix C Example Video Rental Application	27
C.1 DPG Source File	27
C.2 C Output Code	31
Appendix D DPG Source Code	40

D.1 VIDEO.H	41
D.2 VIDEO.C	42
D.3 FILES.H	62
D.4 FILES.C	63
D.5 DPG.C	78

## 1 Introduction

The Database Program Generator (DPG) is a system for creating database software. It is designed to be used by professional programmers and analysts, as opposed to end users; however, it is designed to make the process of constructing and maintaining database software much less time-consuming than it would otherwise be.

Programs created using the system are first written in a fourth-generation language (4GL) which is specifically designed for the purpose. They are then translated into the C language, which is a general purpose programming language available on almost all computer systems. The C code is compiled and linked with some run-time support code to create a stand-alone program which can be run by an end user without access to DPG itself.

DPG is able to create programs which manipulate data consisting of fixed-length records containing textual and numeric information. It is not designed to create programs which handle such data types as graphics, sound, large text files, hypertext etc. However, it is possible to write code to handle these data types in C and link this code in with a DPG application.

The motivation for writing the system is to fill the gap between database management systems (DBMSs) which provide data handling functions, but require the rest of an application program to be written in a conventional programming language, and fourth-generation language packages (4GLs) such as dBase, which are designed to be used by people who are not professional programmers, and which therefore have seriously limited functionality. DPG provides a procedural language with power equivalent to that of a conventional programming language, but provides support for the development of complete application programs, not only the data handling functions.

## 2 Background

A large proportion of the world software market is for systems which maintain data files consisting of fixed-length records containing text and numeric information. Such a system is typically organized as follows:

The data is maintained as a collection of files. Each file contains zero or more records. Each record is composed of one or more fields, each field being a single item of text or numeric information of a fixed length. All the records in a given file are composed of the same fields. An example of this is an employee file for a company, which contains two files: Employees and Departments. Each department has a number, name, and headquarters building; each employee has a number, name and salary. In addition to storing information specific to employees and departments, the database must also store information on how they relate to each other.

## 2.1 Data Models

There are three models for organizing this information: the hierarchical, network and relational models. Supposing there are two relations between entities in this example: an employee can supervise zero or more other employees, and each employee works in exactly one department. This information would be stored in each of the models as follows:

### 2.1.1 The Hierarchical Model

In the hierarchical model, the fundamental relationship between entities is the parent-child relationship, where one file is selected as the parent, and each one is associated with zero or more records from the child file. In this example, we could select Departments as the parent file, and Employees as the child file. The resulting database might then look like this:

This model is simple and efficient, and works well when the data to be represented is hierarchical in nature. However, it has serious limitations when other relationships must be represented; in the above example, there was no direct way of representing the fact that one employee supervises another, although this limitation was overcome by inserting the Supervisor-No field in the Employee file. Furthermore, it is impossible to insert a record for an employee who is not assigned to any department, although this limitation could also be overcome by inserting a dummy Department record. Due to these and other limitations, the hierarchical model is falling out of favour.

### 2.1.2 The Network Model

In the network model, there are two basic data structures: records and sets. A set relates one record to zero or more other records. Our employee database could be represented as follows:

The network model is capable of handling the supervision relationship. However, it was necessary to create a linking (dummy) record type, Supervision, in order to do it. Also, while in actual implementation the data linking one record to another would be stored together with the rest of the record data, such is the number of different links involved that it was necessary here to

draw several separate diagrams to illustrate them all. So the primary disadvantage of the network model is its complexity.

### 2.1.3 The Relational Model

The relational model does not have any physical links between one record and another that are visible to the user (though an implementation may use links internally). Instead, the links are implied by key fields [Codd90]:

Thus, to determine which employees Roberts supervises, we would go through the Employees file and select all those with a Supervisor-No of 675. The relational model is currently the most popular for its conceptual simplicity, the fact that it has a well-developed theoretical foundation and the fact that it has a query language, SQL, which operates on entire blocks of records at a time (query languages for the other data models generally only operate on single records at a time) thereby making the process of constructing complex ad-hoc queries much easier.

For further details of the data models and other database issues, see [Bork80], [Elma89].

## 2.2 Database Management Systems

Originally, database applications were written in ordinary procedural languages, most commonly COBOL. During the 1970s, database management systems (DBMS) were developed. A DBMS is a system accessed from within a procedural language, which handles the storage and maintenance of the data. (Actually the relational and network data models were not developed until after the first DBMSs came on the scene.) There are several advantages to basing an application on a DBMS rather than writing it all in a procedural language:

The application programmer does not need to write the file maintenance code; it is already provided.

Several applications can use the same central database, thus controlling redundancy; in the above example, a payroll application and a project management application could share the employee data. This saves storage space and prevents inconsistency.

A DBMS will provide facilities such as concurrency control and error recovery which often will not be provided in an application written entirely in a procedural language.

Several users can use different applications which provide different views of the same data. For example, a project management application could access all of the

employee data except the salary figures. This is good for security, and may also save training costs by avoiding overloading users with unnecessary information.

If it is desired to change the file format, e.g. to switch to a more efficient indexing method, application programs do not have to be extensively modified.

A DBMS will often provide an ad-hoc query language such as SQL which allows complex queries to be formulated interactively, without the need for writing a program.

(However, since DBMSs generally incur an overhead, there are some applications for which they should not be used, such as a simple single-user application on a system where machine resources are at a premium.)

## 2.3 Fourth Generation Languages

In the 1980s, with the development of the personal computer, there was a demand for tools to simplify the writing of other parts of an application program, primarily the user interface. Packages such as dBase [Pace86] were developed, which allowed users to create simple applications even if they did not have any programming experience; such packages generally have an interactive screen designer, in which users can create input screens by positioning data items with cursor keys or mouse, and also a simplified fourth-generation language (4GL) for performing calculations and updates.

However, since packages like dBase are designed to be usable by someone with no experience in procedural programming languages, they pay a price in flexibility which limits what can be done with them. Complex relationships between files are generally difficult to construct, as are the input screens for such relationships. Such packages also generally require a lot of run-time support; dBase cannot generate stand-alone executable files, and while another package called Clipper can, the compilation process is extremely slow, and the resulting application is large and slow.

## 2.4 A Procedural Application Generator

These disadvantages result from the decision to make the tools usable by non-programmers. There is a need for something in between a DBMS and an interactive 4GL, something which can be used to create an entire application, not just the file handling part, and yet has the power, flexibility and run-time efficiency of a procedural language, to assist programmers in creating applications more easily than they could do with a DBMS.

## 3 Design and Implementation

The drawbacks of an interactive 4GL stem from two sources: the interactive design, and the simplified language for performing updates etc.

The interactive design means that the format of the data in the database is described by meta-data stored in the application program file, and in memory during execution. In an application written in a conventional language, the format of the data is determined by the application code and fixed at compile time. Therefore DPG must adopt this approach, where the application is written in the DPG language and then compiled into an executable file before it can be run. This approach is desirable anyway from the point of view of generating stand-alone applications which the user does not need DPG to run.

In an interactive 4GL package the language is designed to be usable by non-programmers, and is therefore generally not a full procedural language. DPG must use a procedural language with variable assignment, WHILE loops, conditionals etc. just as general-purpose procedural languages do.

The decision was taken to have DPG generate C code as its output, the C code then being compiled with a standard C compiler to generate an executable. There are only two alternatives to generating output in a general-purpose programming language:

- Generate an intermediate code format which is then executed by a run-time interpreter, as many early Pascal compilers did.

- Generate machine code directly.

The first alternative was rejected for the following reasons:

- Interpreting intermediate code at run time incurs a heavy performance overhead.

- The applications generated with such a system could not be run without having access to the run-time interpreter.

- It would be necessary to write a run-time interpreter as well as a compiler to intermediate code, which would increase the work in designing, implementing and maintaining DPG.

- It is desired to enable applications programmers to link in their own code written in a general-purpose language. This would be very difficult if not impossible with run-time interpretation.

The second alternative was rejected for the following reasons:

- DPG would be very difficult to port to systems using a different CPU; it would be necessary to completely rewrite the code generator module.

- The resulting code would probably be less efficient than that generated from using C as intermediate code; although there is some run-time efficiency penalty imposed by



the extra compilation step, this is very small; and the best modern C compilers have had many man-years of effort put into optimizing their code generation, which it would be difficult to duplicate for DPG.

In addition, generating output in a general-purpose language has the following special advantages:

It made DPG very easy to debug, since most bugs in the code generation could be located simply by visual inspection of the output.

It is possible to modify the output code without having to link in external routines.

An application created using DPG can be ported to a system to which DPG itself has not yet been ported.

The only disadvantage is the increased compilation time due to the extra compilation step, and modern C compilers are sufficiently fast as to make this extra time very short.

C was chosen in preference to any other general-purpose language as both the output language and the implementation language for DPG itself for the following reasons:

It has good run-time efficiency.

It has more powerful facilities for handling data structures than all but a few other languages.

It is available on almost all CPUs and operating systems.

Extremely good development tools are available for C.

C is a very popular language and therefore an application programmer using DPG is likely to be able to understand the output code.

The compilers available for C are faster than those available for most other languages.

Had C not been used, Pascal or Modula-2 would probably have been used instead. (Ada would be a good choice, except that it takes a very long time to compile.)

### 3.1 File Structure

The data model used is a subset of the network model (or equivalently, a superset of the hierarchical model). The relational model was considered, but to efficiently implement this model is a complex and difficult undertaking. Furthermore, since DPG uses a procedural language anyway, the ability to use query languages which operate on blocks of records with a

single command is not required. The hierarchical model can be regarded as a subset of the network model; the hierarchical model is too restricted and the full network model is too complex.

DPG provides access to files through indexed key fields; each file may optionally have a single key field, which is required to be unique. (Future versions could have multiple key fields.) This is the main method of access to individual records by application users. A B-tree indexing system is used for these key fields, as B-trees are self-maintaining and provide fast ( $O(\log n)$ ) searching, insertion and deletion.

DPG also supports many-to-one relationships. These differ from the many-to-one relationships of the hierarchical model in that a single record type may have many-to-one relationships with more than one other record type; furthermore, individual records can be entered without being associated with any other record (they have a null for the parent record value). Records may also be unlinked from one parent record and linked onto another.

These relationships are implemented as a doubly linked list which connects all the records associated with a parent record to that parent. Each child record also has a pointer to the parent record, and the parent has pointers to the first and last child records:

Thus, operations such as accessing a parent from a child record and inserting a new child record or deleting an old, can be performed in constant time.

Independently of parent-child pointers, a doubly linked list is also maintained by the system for every file, so that the records of any file may be accessed in the order in which they were entered. (The physical order of records in the file could not be used for this, as records may be deleted from the middle of a file, leaving gaps. The system maintains a separate linked list of free records for each file, and newly inserted records use the first free record, or newly allocated space at the end of the file if there are no free records. Thus the files are self-maintaining.) New records are automatically added to the end of the linked list for the file. For example the sequence:

- Add R1
- Add R2
- Add R3
- Add R4
- Delete R2
- Delete R3
- Add R5

would produce the following file structure:

The application programmer can also access pointers directly, and can therefore implement any file structures not otherwise supported by the system, which may be required.

The many-to-one relationships supported by DPG therefore have most of the power of the sets used by the network model. However, DPG does not directly support many-to-many

relationships, nor repeating groups (fields repeated an arbitrary number of times within a single record occurrence); these must be implemented using many-to-one relationships (or direct manipulation of pointers). Much of the complexity of the full network model is thereby avoided.

A special type of file is also provided, called the "table file". This is simply a file containing a fixed number of records, with no access mechanism other than by record number. Such files are useful for tables of information the implementation of which using indexed files would be complex and inefficient. An example of this is in an accounts system, where different VAT rates may apply to different types of transactions. A table file can be set up, containing say 10 records, each with a single field containing a VAT rate. Each transaction is given a VAT code from 0 to 9, and the application looks up the appropriate VAT rate from the table. When the files are first created, the system initializes all records in the table to zero, after which records can be modified, but not inserted or deleted.

It was necessary to decide whether to use a DBMS for the file handling, or to write a set of routines to do this. It was decided to write the routines, for the following reasons:

For this pilot project, facilities provided by a DBMS such as concurrency control and error recovery which would be needed in a commercial system, were not required.

A DBMS is a general-purpose package and therefore must provide all of the functionality required by any of its users. This tends to impose a penalty in execution speed, and even more in code size, relative to a module written specifically for a particular system.

The requirement to make the system work with an existing DBMS would necessarily influence other design decisions. This would be undesirable for a pilot project whose main purpose is to explore the feasibility and characteristics of this type of system.

Portability might be impaired by using a DBMS, as it would only be possible to port DPG to those machines and operating systems for which a version of the DBMS was available. (Some commercial DBMS packages are available on a wide variety of systems; others only on a few.)

If a full commercial version of DPG were being developed, it would be necessary to decide at that stage whether to switch to a third party DBMS or to continue using a custom-written module.

On disk, the following files exist: each data file is given a name with the extension .DAT; each indexed file also has an index file with the same name as the data file but with the extension .IDX; and there is a system file MISC.DAT which contains all the table files, as well as the start and end pointers for the system linked lists.

An example application for a video rental business was written to demonstrate the system. (Details of the video rental application are given in Appendix C.) The following files are created by this application:

VIDEO.DAT  
VIDEO.IDX  
CUSTOMER.DAT  
CUSTOMER.IDX  
TRANSACTION.DAT  
MISC.DAT

(Under MS-DOS, the file name is truncated to the first eight characters, so TRANSACTION.DAT is actually named TRANSACT.DAT. Also, the file names will be in lower case on operating systems which support this.)

Since a DPG application is separated into two sections, FILES and PROCEDURES (see Appendix B for details), two applications can be written with the same FILES section (perhaps using a macro preprocessor to include a single FILES header file into two or more other source files) but different PROCEDURES sections. This means that it is possible to give different users different views of the same set of data, simply by giving them different application programs.

## 3.2 Application Development

DPG has three components; the compiler, DPG.C, and two run-time modules, VIDEO.C and FILES.C. The system operates as follows: an application is written as a source file with an extension of .DPG. This is compiled into a C source file with the same name as the .DPG file but with an extension of .C. This is then compiled using a C compiler, and linked with the VIDEO and FILES modules to create an executable file. The following commands are used to compile the video rental program, named VRENT, using the MS-DOS version of DPG together with Zortech C v2.1:

```
C:\DPG>DPG VRENT  
Database Program Generator v1.0 by Russell Wallace Apr 23 1992
```

```
C:\DPG>ZTC VRENT VIDEO.OBJ FILES.OBJ  
ztc1b -od:ztc_38K.tmp vrent
```

```
ztc2b d:ztc_38K.tmp -ovrent.obj
```

```
BLINK vrent+video+files/noi;
```

Schematically this process can be represented as:

Total time for this process on a 33-MHz 386 system is approximately 5 seconds. VRENT.DPG is 224 lines long and VRENT.C is 520 lines long, giving a total compilation speed of approximately 2700 lines of DPG code per minute, or 6200 lines of C code per minute, suggesting that the compilation cycle is sufficiently fast for serious development. However, this benchmark is only an approximation; clearly the total time is strongly affected by the speed of the C compiler used, and due to overhead for linking time, the average speed is likely to be

higher for larger applications.

The size of VRENT.EXE is currently 31778 bytes, indicating that the code size overhead imposed by linking with the run-time modules is acceptably low.

Syntax diagrams for the DPG language are given in Appendix A, and a manual is given in Appendix B.

### 3.3 Portability

DPG is currently implemented on MS-DOS using Zortech C v2.1. To port it to another system, the following would be required:

The existence of an ANSI C compiler on the target system.

Recompiling DPG.C and FILES.C on the target system; these should need little or no modification.

Porting VIDEO.C to the target system. This module displays information by writing directly to video memory, as is customary for MS-DOS applications software due to the fact that the MS-DOS implementation of the ANSI terminal codes is too slow. However, this is confined to a few low-level functions (the other functions call these) which could be rewritten quickly and easily.

## 4 Conclusions

On the whole, the project has been successful. The object was to produce a working version of a procedural application generator, a system which would allow professional programmers to create applications software of the same power and efficiency as could be created using a conventional programming language, and yet would allow applications to be created and maintained with relatively little programming effort. While the current version of DPG is not a full commercial system, it was successfully used to create a simple example application, and there seems no reason why much larger programs should not be possible also.

There are disadvantages to the approach used:

DPG is usable only by trained programmers. This was accepted at the start as a necessary consequence of the approach.

A C compiler is an essential link in the development cycle using DPG. Thus, the usefulness of the system would be seriously reduced were a fast, reliable C compiler not available. Fortunately, good C compilers are available on most modern operating systems of commercial importance.

It is difficult to see how an ad-hoc query language such as SQL could be provided for application users, given that DPG is essentially procedural and does not use the relational data model. This is an area where further research could be useful.

An interactive debugging facility would be nice (i.e. a facility whereby application developers could examine the execution of code, the values of variables etc. at run-time). There are problems with providing this, given that the DPG code is translated into C; use of existing debugging programs will only enable examination of the C code at run-time. This is better than nothing, but not ideal. (The situation is reminiscent of the early debuggers provided with C compilers, which could only display assembly language code at run time.) Again, further research in this area could be useful.

The most difficult decisions to make were the decisions to use a subset of the hierarchical data model rather than the relational model, and to write a set of database routines rather than use an existing DBMS package. It is interesting to speculate how the system would look if a relational DBMS package had been used. Provided that the procedural design of DPG could have been matched to a relational DBMS, which is fundamentally not procedural in nature (the classic "impedance mismatch" problem), a number of advantages would have accrued: facilities such as error handling and concurrency control would have been automatically provided, as would an SQL, and any other features supplied by the DBMS, e.g. use of alternative indexing methods. (There would also have been disadvantages; see section 3.1.) Other than this, it is hard to see how any of the major design decisions could have been made differently.

## 5 Future Work

The following features are not yet implemented in the current version of DPG:

Report output to printer or disk. Currently it is only possible to output information to the screen; commands to open, output information to, and close a printer channel are needed.

Linking in C code. Facilities to link in external routines and insert inline C code are required; currently such code must be inserted manually into the output file.

In order to expand the current pilot project into a full commercial system, the following issues would need to be addressed:

Concurrency. This is required for multiuser database applications. It could be provided either by switching to a third party database management system for file handling, or else by providing record locking or other concurrency control mechanisms in the FILES module.

Error Recovery. This is generally required for large-scale database applications, and normally implemented in the form of a log of all database updates, which is

physically written to disk before an update is committed, and from which the database can therefore be reconstructed in the event of a power failure. Again this could be provided either by a third party DBMS or by implementing it in the FILES module.

**Alternative Indexing Schemes.** Indexing is currently performed using B-trees, which are highly efficient for most applications. However, commercial DBMS packages and 4GLs often provide a choice of indexing schemes, because the optimal indexing scheme can vary from one application to another. This would be a desirable feature for a commercial version of DPG. Again this could be provided either by a third party DBMS or within DPG.

**Compiler Error Checking.** The DPG currently performs syntax checking of the input file, but little in the way of semantic checking; such things as assignment of an integer to a string variable, or use of a C keyword as a variable name, are generally left to the C compiler to detect. This is not desirable for serious work.

**Multiple Source Files.** Compilers for C and other general-purpose programming languages generally provide support for the writing of a program as several source files, which are combined into a single executable at link time. For large programs, this shortens compile time, makes it easier to divide a large project among members of a team, and helps to avoid any limits on such things as code or data segment size, which may be imposed by the compiler or operating system. DPG does not currently support this, but its implementation would enhance the usefulness of DPG for large applications.

The following features, while not essential, would add to the usefulness of DPG:

**An interactive screen designer.** This would speed up the process of designing input screens for simple file maintenance and the like, by allowing the user to manually position data items. (More complex input screens would still need to be constructed using the DPG language.) A default screen generator would also be useful, to automatically generate screens for the maintenance of data files; these could then be modified by the application programmer. The screens, once created, would be written out in DPG language form.

**A query language.** It is possible to create reports in the DPG language, but this facility is for use by the application programmer, not the end user. It would be useful for an end user to be able to construct ad-hoc queries. Since DPG does not use the relational data model, a query language like SQL cannot be employed directly; however, it should be possible to design a non-procedural language interpreter of some type for this purpose. Further research on this problem is indicated.

## Bibliography

[Bell84]

Bell DA, Database Performance, Pergamon Infotech (1984).

[Bork80]

Borkin SA, Data Models, MIT Press (1980).

[Codd90]

Codd EF, Edgar F, The Relational Model for Database Management, Addison-Wesley (1990).

[Elma89]

Elmasri R, Navathe SB, Fundamentals of Database Systems, Addison-Wesley (1989).

[Fros84]

Frost RA, Database Management Systems, Granada (1984).

[Gonn91]

Gonnet GH, Baeza-Yates R, Handbook of Algorithms and Data Structures in Pascal and C, Addison-Wesley (1991).

[Pace86]

De Pace MM, The dBase Programming Language, Collins (1986).

## Appendix A DPG Language Grammar

The grammar for the DPG language is presented in a modified Backus-Naur Form (see [Gonn91] for details). The following elements are used:

### A.1 Modified BNF

lowercase\_word

UPPERCASE\_WORD

.

any printable ASCII character

'symbol'

literal symbol e.g. '|' means '|' not 'or'



::=

consists of

[]

optional

\*

zero or more occurrences of

+

one or more occurrences of

|

or

[a-z]

a to z inclusive

()

priority of evaluation

## A.2 DPG Language

program ::=

variable ::=

int\_var ::=

INT integer symbol

float\_var ::=

date\_var ::=

str\_var ::=

STR integer symbol

ptr\_var ::=

PTR symbol

file ::=

symbol '{' variable\* '}'

[ON '{' symbol\* '}']

[TABLE integer]  
procedure ::=  
statement ::=  
  
symbol '=' expr |  
symbol '.' symbol '=' expr |  
symbol |  
EDIT lvalue expr expr |  
SELECT symbol expr expr directions |  
FIND symbol expr |  
READ symbol expr |  
WRITE symbol [expr] |  
CREATE integer integer symbol directions |  
CREATE symbol |  
DELETE symbol |  
LINK symbol symbol |  
UNLINK symbol symbol |  
SCROLL symbol symbol expr symbol symbol expr expr expr |  
GOTO symbol |  
BREAK |  
CONTINUE |  
RETURN |  
IF expr statement [ELSE statement] |  
WHILE expr statement |

DO statement WHILE expr |  
SWITCH expr '{ symbol\* }' |  
PRINTINT expr expr expr [expr] |  
PRINTFLOAT expr expr expr [expr expr] |  
PRINTDATE expr expr expr |  
PRINTSTR expr expr expr [expr] |  
SPRINTINT expr expr [expr] |  
SPRINTFLOAT expr expr [expr expr] |  
SPRINTDATE expr expr |  
CURSOR expr expr |  
PAUSE |  
GETKEY expr |  
STRCPY expr expr [expr]  
directions ::=  
direction ::=  
dir\_name ::=  
  
UP |  
DOWN |  
LEFT |  
RIGHT |  
PGUP |  
PGDN |  
HOME |

END |

INS |

DEL

expr ::=  
and\_expr ('||' and\_expr)\*

and\_expr ::=

eq\_expr ::=

rel\_expr ::=

math\_expr ::=

term ::=  
factor (('\*' | '/' | '%') factor)\*

factor ::=  
'-' factor |

integer |

float |

string |

(' expr ') |

lvalue |

ATODATE expr |

ATOF expr |

ATOI expr |

DATEDAY expr |

DATEMONTH expr |

DATEYEAR expr |

MAKEDATE expr expr expr |

STRCMP expr expr [expr]

lvalue ::=  
symbol | symbol '.' symbol

symbol ::=  
alpha alphanum\*

integer ::=  
digit+

float ::=  
digit\* '.' digit+ | digit+ '.' digit\*

alpha ::=  
[a-z] | [A-Z] | \_

digit ::=  
[0-9]

alphanum ::=

string ::=  
''' .\* '''

## Appendix B DPG Language User Manual

### B.1 Introduction

A DPG program is regarded as consisting of a stream of "tokens", separated by whitespace (i.e. space, tab or newline characters). Comments are also treated as whitespace; comments are started and ended by the symbols `/*` and `*/` (these may not be nested). There are five types of token:

#### B.1.1 Keywords

The following are the reserved keywords of DPG, they may not be used otherwise:

ATODATE, ATOF, ATOI, BREAK, CONTINUE, CREATE, CURSOR, DATE, DATEDAY, DATEMONTH, DATEYEAR, DEL, DELETE, DO, DOWN, EDIT, ELSE, END, ESC, FILES, FIND, FLOAT, GETKEY, GOTO, HOME, IF, INDEX, INS, INT, LEFT, LINK, MAKEDATE,

ON, PAUSE, PGDN, PGUP, PRINTDATE, PRINTFLOAT, PRINTINT, PRINTSTR, PROCEDURES, PTR, READ, RETURN, RIGHT, SCROLL, SELECT, SPRINTDATE, SPRINTFLOAT, SPRINTINT, STR, STRCMP, STRCPY, SWITCH, TABLE, UNLINK, UP, VAR, WHILE, WRITE

For clarity in this manual, DPG keywords will be given in upper case, however DPG is not case sensitive, and upper case, lower case or a mixture of the two can be used in writing DPG programs.

## B.1.2 Identifiers

An identifier is any non-keyword token consisting of a letter followed by zero or more alphanumeric characters. An underscore is regarded as a letter in this context. Identifiers may be any length, but only the first 31 characters are significant.

An identifier can denote one of the following things: a local variable, a global variable, a file, a procedure, a field within a file, or a label. In general, no two things may have the same name, even if they are different types of things. There are exceptions to this. Local variables only exist within the procedure in which they are defined. If a local variable and a global variable have the same name, then in the procedure in which the local variable is defined, the local variable will have precedence. Two fields in different files may have the same name. And label names are completely independent of names for other types of things; the only restriction is that a label name may not be a DPG keyword, and the same label may not be defined twice within one procedure.

While DPG is not case sensitive, C is, so in the output C code, all identifiers are converted to lower case, except for labels which in accordance with C convention are converted to upper case. This is only relevant if you intend to modify the C code or link it with your own code.

In addition to the DPG keywords, the following are reserved words in C and therefore may not be used as identifiers in DPG (except as labels; these are output in upper case and are therefore considered distinct from lower case keywords by the C compiler):

auto, break, case, char, const, continue, default, do, double, else, entry, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

Note that some C keywords are also DPG keywords. Use of a C keyword which is not a DPG keyword as an identifier will not be detected as an error by the current version of the DPG compiler, instead the C compiler will detect this.

## B.1.3 Constants

A sequence of one or more digits is regarded as an integer constant. A sequence of one or more

digits containing a dot is regarded as a floating-point constant.

## B.1.4 Strings

Any sequence of printable ASCII characters surrounded by double quotes is a string constant, e.g. "This is a string". The quotes are not regarded as part of the string.

## B.1.5 Other Symbols

These include operators, punctuators etc. e.g. +, -, =, {, }.

## B.2 Program Composition

A DPG program consists of the following in this order: an optional list of global variable declarations, the word FILES, zero or more file definitions, the word PROCEDURES and zero or more procedure definitions.

## B.3 Variable Declarations

A variable declaration consists of one of the following:

INT length name

This is an integer variable which can hold positive or negative values of up to length digits. INT variables are implemented as signed char, short or long, whichever is the smallest needed to hold the indicated number of digits (this may vary depending on the target system).

FLOAT length decimals name

This is a floating-point variable, which will be displayed in a format having length digits before the decimal point and decimals digits after it. Regardless of the values of length and decimals, FLOAT variables are always implemented as double (which on most systems gives about 14 significant digits of precision).

DATE name

This is a variable which stores a date; it is currently implemented as unsigned short. Dates may be compared just as numeric expressions can; a DATE variable may be resolved into its component parts using the DATEDAY, DATEMONTH and DATEYEAR operators, and may be put back together again using the MAKEDATE operator. Dates are displayed in the form DD.MM.YY.

STR length name

This is a string variable which stores length characters. (In the C code, null terminated strings are not used; instead, the length of each string is fixed at compile time. This means that an extra byte for the null terminator is not required.)

PTR name

This is a variable which stores a pointer to a record in a file. In the current implementation, a negative value denotes a null pointer, and a non-negative value denotes a byte offset within a file; however, this may change in future versions, therefore it is preferable to perform no operations on PTR variables other than assignment to each other and testing for negative values. PTR variables are currently implemented as "long". (Remember that records in table files are accessed using integer record numbers, not by pointers.)

## B.4 File Definitions

A file definition results in the creation of a C structure definition, space in memory for one record, a null record for initializing new records, and variables for keeping track of the file. For example, in the video rental package, the definition of the file video resulted in the following:

```
typedef struct { ... } video;
video blankvideo = { ... };
video video;
int videofile;
long videoptr;
int videoindexfile;
```

For a record on disk to be accessed, it must first be read into memory; one way to do this is with the READ command. Space is reserved in memory for a single video record; its fields may be accessed using the notation video.fieldname as in C. The blankvideo variable contains null values to be used in the creation of new records (i.e. it is a video record with all string fields set to spaces, all numeric fields set to zero and all pointer fields set to null). It can not be accessed within DPG.

The variable videoptr can be accessed from DPG; it contains a pointer to the place on disk where the record currently in memory belongs. This is so that when the record in memory is modified, DPG will know where to write it back. It is not normally necessary to manually access this variable, as the READ command automatically sets it and the WRITE command automatically uses it.

The application automatically tests for the existence of the files VIDEO.DAT and VIDEO.IDX on startup, and creates them if they do not exist. (The file VIDEO.IDX would not be created if there was no index field, and the file VIDEO.DAT would not be created if video was a table file;



the data would be stored in MISC.DAT.)

A file definition consists of a name and then the following in the given order:

### B.4.1 Fields

The fields of a file are given as a list of variable declarations surrounded by curly brackets.

### B.4.2 Index

A file can optionally have an index field, which must be a string field. This is not defined with the other fields, but after them as follows:

INDEX name length

The index field may be accessed in the same way as the other fields, except that once a record is created, its index field value may not be altered.

### B.4.3 Linked On

A file may be linked on one or more other files (i.e. have a many-to-one relationship with them). This is specified by the word ON, followed by a list of the names of other files, surrounded by curly brackets.

When a file is linked on another one, the following fields of type PTR will be automatically created: othernameptr, othernamenext, othernameprev (where othername is the name of the other file). These fields may be accessed in the same way as ordinary fields. They contain: a pointer to the record which the current record is linked on, a pointer to the next record linked on the same master record, and a pointer to the same record linked on the previous master record, respectively. A null value for the ptr field indicates that the current record is not linked on any master record at the moment; null values for the next and prev fields indicate the end and start of the linked list respectively. It is not normally necessary to maintain these fields manually, as this is done by the file maintenance functions.

When another file is linked on the current file, the current file will have the following extra fields automatically created: othernamefirst, othernamelast. These fields contain pointers to the start and end of the linked list of records linked on the current record. They are otherwise similar to the ptr, next and prev fields which the other file will have. (See Appendix C for examples of this.)

## B.4.4 Table

If the file is a table, this is specified as follows:

TABLE records

where records is the fixed number of records in the file; this can be any integer greater than zero. A table file may not have an index field, nor may it be linked on any other files, nor may it have any files linked on it.

## B.5 Procedures

A DPG procedure translates into a C function with no arguments returning type void. A procedure definition consists of the following:

A name.

An optional menu item name. If given this should consist of a string constant containing the name of a menu item associated with this procedure. An example from the video rental application is "Videos!Add a Video". This means that the procedure is called when the user selects the option "Add a Video" from the submenu "Videos" (the ! character delimits submenu names; up to 20 levels of submenus are allowed). If no menu item name is given, the procedure can only be called from within another procedure and is not directly accessible to the user.

A list of zero or more local variables. Note that unlike C, DPG requires local variables to be declared outside the block of statements for a procedure. Also, local variables can only be declared at the start of the procedure, not within a nested statement block.

Zero or more statements surrounded by curly brackets.

## B.6 Statements

A statement consists of one of the following:

Zero or more statements surrounded by curly brackets.

A colon followed by a label. (This is the opposite of C, where the colon comes after the label.)

An assignment statement; this is a variable name followed by the = symbol, followed by an expression. (In all cases where a variable name is permitted, this can be the name of a local or global variable, or a file name followed by a dot, followed by the name of a field within that file.)

The name of a procedure. This designates a call to the indicated procedure. The called procedure must have already been defined, or be the current procedure; this means that recursion is allowed, but two procedures cannot be mutually recursive.

#### EDIT x y variable directions

This allows the user to type in a new value for the indicated variable. The cursor will be positioned at location x,y on the screen (the top left hand corner of the screen is 0,0). A negative y coordinate indicates that the current cursor y coordinate is to be used; this is used in the SCROLL command. Directions indicates actions to be taken depending on what direction key the user presses; this is discussed in B.7.

#### SELECT filename x y directions

This positions the cursor at location x,y and allows the user to type in a value for the index field of the indicated file (obviously, this statement cannot be used for non-indexed files). Unless the user has pressed the ESC key, the system will attempt to locate the indicated record and read it into memory. If the indicated record does not exist, the user will be asked to key a new value.

#### FIND filename expression

This attempts to find the record in the indicated file with the index field value given by expression, and read it into memory. If no such record exists, the file pointer variable will be set to null (a negative value); this should be tested to determine whether the search succeeded.

#### READ filename expression

This reads the record indicated by expression from the indicated file into memory and sets the file pointer variable; expression should be of type INT for a table file and PTR for other files. One use for this command is for stepping through a linked list, where expression will be the pointer to the next record. The result of attempting to READ an invalid record is not defined.

#### WRITE file

This writes the record from file currently in memory back to disk, using the file pointer variable. The result of using this command with an invalid file pointer is not defined.

#### WRITE file expression

This writes the record from file currently in memory back to disk. File must be a table file; expression indicates the record number. The result of using this command with an invalid record number is not defined.

#### CREATE x y filename directions

This positions the cursor at x,y and allows the user to key a value for the index field of the indicated file. If the user pressed ESC, the file pointer variable is set to null; otherwise, a record

is created with all string fields other than the index field set to spaces, all numeric fields set to zero and all pointer fields set to null, and the file pointer value is set accordingly.

`CREATE filename`

This creates a new record in the indicated file. This form should be used for non-indexed files.

`DELETE filename`

This deletes the record currently in memory from the indicated file (it requires that the file pointer variable be correctly set). It is up to the application programmer to ensure that any records linked onto the current record are either unlinked or themselves deleted before the `DELETE` command is used for the current record; however, any linkage from the current record is automatically handled.

`LINK fromfile tofile`

This links the record currently in memory from `fromfile` to the record currently in memory from `tofile` (a many-to-one relationship from `fromfile` to `tofile` must have been defined in the `FILES` section). The `LINK` command cannot be used on a record that is currently linked onto one record to link it to another record; it must be `UNLINKed` before being re-`LINKed`.

`UNLINK fromfile tofile`

This unlinks the record currently in memory from `fromfile` to the record currently in memory from `tofile`. The result of using the `UNLINK` command where no such link already exists is not defined.

`SCROLL fromfile tofile nfields fshow fedit x y width height`

This allows the user to edit a list of records associated with another record. `FROMFILE` must be linked on `TOFILE`. There must be a record currently in memory from `TOFILE`, but not from `FROMFILE`. `X` and `y` are the coordinates of the top left corner of the edit window. `Width` and `height` are the width and height of this window. `Nfields` is the number of fields which will be displayed across the screen.

The `SCROLL` command can be used in one of two ways. If a zero is given for `fedit`, then `fshow` is the name of a procedure which will display a record from `fromfile`. `fshow` must be written to display fields from the `fromfile` record currently in memory, at the current `y` coordinate; this is done by using `print` statements within `fshow` with negative `y` coordinates supplied. Before the `SCROLL` command is used, a title bar should be placed above the display window with the names of the fields. The result of all this will be to allow the user to scroll up and down the list of `fromfile` records associated with the current `tofile` record. (When `SCROLL` is used in display-only mode, the `nfields` parameter is not relevant and a zero may be given for it.) Note that `fshow` must have been defined before the function in which the `SCROLL` command is used.

In edit mode, the other parameters are as before, but `fedit` is the name of a function which will

edit the fromfile record currently in memory, at the current y coordinate; again, this is done by supplying -1 as the y coordinate to the EDIT commands within fedit. Fedit should also handle cursor movement, as in the following example from VRENT.DPG:

```
enter_transactions_edit
{
  switch fieldno { TITLE LOAN_DATE TYPE }
:TITLE
  edit transaction.title 0 (-1)
    { return esc up down right pgup pgdn home end ins del }
  return
:LOAN_DATE
  edit transaction.transaction_date 30 (-1)
    { return esc up down left right pgup pgdn home end ins del }
  return
:TYPE
  edit transaction.type 50 (-1)
    { return esc up down left pgup pgdn home end ins del }
  return
}
```

The SCROLL function automatically writes changed data in memory back to disk. Where it is not desired to allow the user to use the INSERT and DELETE keys to create and delete records, INS and/or DEL should be omitted from the directions supplied to the edit commands.

**GOTO label**

This transfers program control to the statement after the indicated label.

**BREAK**

This jumps out of the enclosing WHILE or DO ... WHILE loop.

**CONTINUE**

This jumps to the end of the enclosing WHILE or DO ... WHILE loop.

**RETURN**

This returns from the current procedure.

**IF expression statement [ELSE statement]**

This evaluates the expression (which should be numeric); if the result is non-zero, then the statement is executed; otherwise, if the ELSE part is specified, then the statement after the word ELSE is executed.

WHILE expression statement

This executes the statement as long as the expression has a non-zero value.

DO statement WHILE expression

This is similar to a WHILE loop, except that the statement is always executed once before the expression is tested for the first time.

SWITCH expression { labels }

This is less general than the C switch statement; it evaluates the expression, which should be numeric. If the result is zero, program control is transferred to the statement after the first label. If the result is one, program control is transferred to the statement after the second label, etc. For example,

```
switch fieldno { TITLE LOAN_DATE TYPE }
```

jumps to TITLE, LOAN\_DATE or TYPE depending on whether the value of the variable fieldno is 0, 1 or 2 respectively. (If the variable is outside this range, no action is taken.) The primary use of this statement is in conjunction with SCROLL.

PRINTINT x y expression [length]

Displays an integer expression on the screen at coordinates x,y. If length is given, this value will be used, otherwise DPG will determine the value (e.g. if expression is a variable, the length declared for the variable will be used).

PRINTFLOAT x y expression [length decimals]

Displays a floating point expression on the screen at coordinates x,y. If length and decimals are given, these values will be used, otherwise DPG will determine the values.

PRINTDATE x y expression

Displays a date expression on the screen at coordinates x,y.

PRINTSTR x y expression [length]

Displays a string expression on the screen at coordinates x,y.

SPRINTINT string expression [length]

Converts an integer expression to a string and copies this into a string variable. The string variable should be at least length characters long; if it is longer, extra characters will not be affected.

SPRINTFLOAT string expression [length decimals]

Converts a floating-point expression to a string and copies this into a string variable.

SPRINTDATE

Converts a date expression to a string and copies this into a string variable.

CURSOR x y

Positions the hardware cursor at location x,y.

PAUSE

Waits for the user to press a key.

GETKEY string

Waits for the user to press a key, and places the resulting value into the first character of a string variable.

STRCPY string expression [length]

Assigns a string variable to a string expression. (The C code for this uses the memcpy() function.)

## B.7 Directions

All statements which allow the user to key a value for a variable require a list of directions to be supplied. This consists of either a single direction, or zero or more directions surrounded by curly brackets. A direction consists of a direction keyword optionally followed by the = symbol and a label. An example is:

```
edit video.copies 30 7 { return esc=END up=CATEGORY down }
```

This means that having keyed a value for the field video.copies, the user can press the RETURN, ESC, UP or DOWN keys. If the RETURN or DOWN keys are pressed, execution continues to the next statement. If the ESC key is pressed, execution is transferred to the statement following the END label. If the UP key is pressed, execution is transferred to the statement following the CATEGORY label. A full list of available direction keywords is as follows:

RETURN, ESC, UP, DOWN, LEFT, RIGHT, PGUP, PGDN, HOME, END, INS, DEL

## B.8 Expressions

An expression consists of a variable, a constant number or string, the result of a function call or one of the following operators applied to one or two expressions:

```
!   -  
*   /   %  
+   -  
<   <=   >   >=  
==  !=  
&&  
||
```

The operators are listed in order of precedence, for example \* and / have higher priority than + and -. An expression surrounded by brackets is also an expression, and this is used to change operator precedence.

The binary arithmetic operators are + (addition), - (subtraction), \* (multiplication), / (division) and % (remainder). Division performed on two integers always rounds the result down; use the % operator to obtain the remainder. Note that % cannot be used on expressions of type FLOAT. Also, the unary minus operator is considered different from the subtraction operator and has a higher precedence, so that - 4 \* 6 is the same as (- 4) \* 6.

The binary relational operators are < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to) and != (not equal to). Expressions using these evaluate to either 0 (FALSE) or 1 (TRUE). They can be used on dates, but cannot be used on strings, the STRCMP function must be used instead.

&&, || and ! are the logical AND, OR and NOT operators respectively. These take zero or non-zero results and evaluate to either 0 or 1.

The following functions are provided:

ATODATE string

Converts a string expression of the form "DD.MM.YY" or "DD/MM/YY" to a date.

ATOF string

Converts a string expression to a floating-point number.

ATOI string

Converts a string expression to an integer.

DATEDAY date



Obtains the day number (1 to 31) for a date.

`DATEMONTH date`

Obtains the month number (1 to 12) for a date.

`DATEYEAR date`

Obtains the year number (0 to 99) for a date.

`MAKEDATE day month year`

Converts the day, month and year values (which should be integers) to a date.

`STRCMP string1 string2 [length]`

Compares two strings and returns an integer value which is negative if `string1 < string2`, positive if `string1 > string2` and zero if `string1 == string2`. This function is needed because the relational operators cannot be used directly on strings. The comparison is performed without regard to the case of the strings, i.e. `STRCMP "FRED" "fred" == 0`. (The C code for this uses the `memcmp()` function.) If `length` is given, this value will be used, otherwise DPG will determine the value.

Note that unlike C, no brackets are required around function arguments.

The functions have lower precedence than the operators, for example `MAKEDATE 31 12 90 + 2` is the same as `MAKEDATE 31 12 (90 + 2)`.

In general, different types cannot be mixed in expressions. `INT` and `FLOAT` types can be mixed, the `INT` is converted to a `FLOAT` before the operation is performed. An integer can be added to a string to obtain part of the string, e.g. `STRCMP "fred" + 2 "ed" 2 == 0`. (This is one situation where the `length` parameter for commands and functions such as `STRCMP` should be provided.) `PTR` expressions can be mixed with numeric expressions, except that the resulting value may be implementation dependent, so that it is best to avoid using `PTR` values for anything other than testing for null pointers (i.e. negative values).

The current version of DPG does not check for invalid mixing of expression types, so that such an error is detected only by the C compiler. Care should therefore be taken in this area.

## Appendix C Example Video Rental Application

### C.1 DPG Source File

files

```
video
{
  str 20 category
  int 2 copies
  int 2 in_stock
  float 2 2 rent
  date release_date
}
index title 20
```

```
customer
{
  str 20 full_name
  str 20 address_1
  str 20 address_2
  str 20 address_3
}
index account_name 20
```

```
transaction
{
  str 20 title
  date transaction_date
  str 1 type
}
on { customer }
```

procedures

```
wait
{
  printstr 30 20 "Press any key to continue"
  cursor 56 20
  pause
}
```

```
show_video
{
  printstr 10 6 "Category"
  printstr 10 7 "Copies"
  printstr 10 8 "Copies in Stock"
  printstr 10 9 "Rental Charge"
  printstr 10 10 "Release Date"

  printstr 30 5 video.title
  printstr 30 6 video.category
```

```

    printint 30 7 video.copies
    printint 30 8 video.in_stock
    printfloat 30 9 video.rent
    printdate 30 10 video.release_date
}

alter_video
{
    show_video
:CATEGORY
    edit video.category 30 6 { return esc=END down }
:COPIES
    edit video.copies 30 7 { return esc=END up=CATEGORY down }
:IN_STOCK
    edit video.in_stock 30 8 { return esc=END up=COPIES down }
:RENT
    edit video.rent 30 9 { return esc=END up=IN_STOCK down }
:RELEASE_DATE
    edit video.release_date 30 10 { return esc up=RENT }
:END
    write video
}

add_video "Videos!Add a Video"
{
    printstr 10 5 "Title"
    create 30 5 video { return esc=CANCEL }
    alter_video
    return
:CANCEL
    delete video
}

edit_video "Videos!Edit a Video"
{
    printstr 10 5 "Title"
    select video 30 5 { return esc=CANCEL }
    alter_video
:CANCEL
}

display_video "Videos!Display a Video"
{
    printstr 10 5 "Title"
    select video 30 5 { return esc=CANCEL }
    show_video
    wait
}

```

```

:CANCEL
}

remove_video "Videos!Remove a Video"
str 1 sure
{
  printstr 10 5 "Title"
  select video 30 5 { return esc=CANCEL }
  show_video
  printstr 30 20 "Are you sure? (Y/N) "
  cursor 50 20
  getkey sure
  if !strcmp sure "Y"
    delete video
:CANCEL
}

show_customer
{
  printstr 10 6 "Full Name"
  printstr 10 7 "Address"

  printstr 30 5 customer.account_name
  printstr 30 6 customer.full_name
  printstr 30 7 customer.address_1
  printstr 30 8 customer.address_2
  printstr 30 9 customer.address_3
}

alter_customer
{
  show_customer
:FULL_NAME
  edit customer.full_name 30 6 { return esc=END down }
:ADDRESS_1
  edit customer.address_1 30 7 { return esc=END up=FULL_NAME down }
:ADDRESS_2
  edit customer.address_2 30 8 { return esc=END up=ADDRESS_1 down }
:ADDRESS_3
  edit customer.address_3 30 9 { return esc up=ADDRESS_2 }
:END
  write customer
}

add_customer "Customers!Add a Customer"
{
  printstr 10 5 "Account Name"

```

```

    create 30 5 customer { return esc=CANCEL }
    alter_customer
    return
:CANCEL
    delete customer
}

edit_customer "Customers!Edit a Customer"
{
    printstr 10 5 "Account Name"
    select customer 30 5 { return esc=CANCEL }
    alter_customer
:CANCEL
}

display_customer "Customers!Display a Customer"
{
    printstr 10 5 "Account Name"
    select customer 30 5 { return esc=CANCEL }
    show_customer
    wait
:CANCEL
}

remove_customer "Customers!Remove a Customer"
str 1 sure
{
    printstr 10 5 "Account Name"
    select customer 30 5 { return esc=CANCEL }
    show_customer
    printstr 30 20 "Are you sure? (Y/N) "
    cursor 50 20
    getkey sure
    if !strcmp sure "Y"
        delete customer
:CANCEL
}

enter_transactions_show
{
    printstr 0 (-1) transaction.title
    printdate 30 (-1) transaction.transaction_date
    printstr 50 (-1) transaction.type
}

enter_transactions_edit
{

```

```

    switch fieldno { TITLE LOAN_DATE TYPE }
:TITLE
    edit transaction.title 0 (-1) { return esc up down right pgup pgdn home end ins del }
    return
:LOAN_DATE
    edit transaction.transaction_date 30 (-1) { return esc up down left right pgup pgdn home end
ins del }
    return
:TYPE
    edit transaction.type 50 (-1) { return esc up down left pgup pgdn home end ins del }
    return
}

```

```

enter_transactions "Transactions!Enter Transactions"
{
    printstr 10 5 "Account Name"
    select customer 30 5 { return esc=CANCEL }
    show_customer
    printstr 0 11 "Film Title"
    printstr 30 11 "Transaction Date"
    printstr 50 11 "Type (L/R)"
    scroll transaction customer 3 enter_transactions_show enter_transactions_edit 0 13 80 12
:CANCEL
}

```

```

display_transactions "Transactions!Display Transactions"
{
    printstr 10 5 "Account Name"
    select customer 30 5 { return esc=CANCEL }
    show_customer
    printstr 0 11 "Film Title"
    printstr 30 11 "Transaction Date"
    printstr 50 11 "Type (L/R)"
    scroll transaction customer 3 enter_transactions_show 0 0 13 80 12
:CANCEL
}

```

## C.2 C Output Code

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>
#include "video.h"

```

```

#include "files.h"

short fieldno;

typedef struct
{
    long next;
    long prev;

    long customerptr;
    long customernext;
    long customerprev;

    char type[1];
    unsigned short transaction_date;
    char title[20];
} transaction;

transaction blanktransaction =
{
    -1,
    -1,

    -1,
    -1,
    -1,

    '',
    BLANKDATE,
    '','',,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
};

transaction transaction;
int transactionfile;
long transactionptr;

#define transactionmiscoeff 0

typedef struct
{
    long next;
    long prev;

    char account_name[20];

    long transactionfirst;
    long transactionlast;
}

```

```
char address_3[20];
char address_2[20];
char address_1[20];
char full_name[20];
} customer;
```

```
customer blankcustomer =
{
  -1,
  -1,

  ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',

  -1,
  -1,

  ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
  ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
  ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
  ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
};
```

```
customer customer;
int customerfile;
long customerptr;
int customerindexfile;
```

```
#define customermiscoff 12
```

```
typedef struct
{
  long next;
  long prev;

  char title[20];

  unsigned short release_date;
  double rent;
  signed char in_stock;
  signed char copies;
  char category[20];
} video;
```

```
video blankvideo =
{
  -1,
```



```

-1,
',,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
BLANKDATE,
0,
0,
0,
',,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
};

video video;
int videofile;
long videoptr;
int videoindexfile;

#define videomiscoff 28

void wait (void)
{
    printstr (30,20,"Press any key to continue",25);
    cursor (56,20);
    getkey ();
}

void show_video (void)
{
    printstr (10,6,"Category",8);
    printstr (10,7,"Copies",6);
    printstr (10,8,"Copies in Stock",15);
    printstr (10,9,"Rental Charge",13);
    printstr (10,10,"Release Date",12);
    printstr (30,5,video.title,20);
    printstr (30,6,video.category,20);
    printint (30,7,video.copies,2);
    printint (30,8,video.in_stock,2);
    printfloat (30,9,video.rent,2,2);
    printdate (30,10,video.release_date);
}

void alter_video (void)
{
    show_video ();
}

CATEGORY:
editstr (video.category,20,30,6,DF_RETURN | DF_ESC | DF_DOWN);
switch (dir)

```

```
{
    case DIR_ESC:
        goto END;
}
```

COPIES:

```
editbyte (&video.copies,2,30,7,DF_RETURN | DF_ESC | DF_UP | DF_DOWN);
switch (dir)
{
    case DIR_ESC:
        goto END;
    case DIR_UP:
        goto CATEGORY;
}
```

IN\_STOCK:

```
editbyte (&video.in_stock,2,30,8,DF_RETURN | DF_ESC | DF_UP | DF_DOWN);
switch (dir)
{
    case DIR_ESC:
        goto END;
    case DIR_UP:
        goto COPIES;
}
```

RENT:

```
editfloat (&video.rent,2,2,30,9,DF_RETURN | DF_ESC | DF_UP | DF_DOWN);
switch (dir)
{
    case DIR_ESC:
        goto END;
    case DIR_UP:
        goto IN_STOCK;
}
```

RELEASE\_DATE:

```
editdate (&video.release_date,30,10,DF_RETURN | DF_ESC | DF_UP);
switch (dir)
{
    case DIR_UP:
        goto RENT;
}
```

END:

```
Write (videofile,videoptr,&video,sizeof video);
}
```

```

void add_video (void)
{
    printstr (10,5,"Title",5);
    video = blankvideo;
    createindex (videofile,&videoptr,&video,sizeof
video,videomiscoff,video.title,20,videoindexfile,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    alter_video ();
    return;

CANCEL:
    deleteindex (videofile,videoptr,&video,videomiscoff,video.title,20,videoindexfile);
}

void edit_video (void)
{
    printstr (10,5,"Title",5);
    select (videofile,videoindexfile,&videoptr,&video,sizeof
video,video.title,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    alter_video ();

CANCEL::
}

void display_video (void)
{
    printstr (10,5,"Title",5);
    select (videofile,videoindexfile,&videoptr,&video,sizeof
video,video.title,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    show_video ();
    wait ();

CANCEL::
}

```

```

}

void remove_video (void)
{
    char sure[1];

    printstr (10,5,"Title",5);
    select (videofile,videoindexfile,&videoptr,&video,sizeof
video,video.title,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    show_video ();
    printstr (30,20,"Are you sure? (Y/N) ",20);
    cursor (50,20);
    sure[0] = getkey ();
    if (!memicmp (sure,"Y",1))
        deleteindex (videofile,videoptr,&video,videomiscoff,video.title,20,videoindexfile);

CANCEL::
}

void show_customer (void)
{
    printstr (10,6,"Full Name",9);
    printstr (10,7,"Address",7);
    printstr (30,5,customer.account_name,20);
    printstr (30,6,customer.full_name,20);
    printstr (30,7,customer.address_1,20);
    printstr (30,8,customer.address_2,20);
    printstr (30,9,customer.address_3,20);
}

void alter_customer (void)
{
    show_customer ();

FULL_NAME:
    editstr (customer.full_name,20,30,6,DF_RETURN | DF_ESC | DF_DOWN);
    switch (dir)
    {
        case DIR_ESC:
            goto END;
    }
}

```

```

ADDRESS_1:
    editstr (customer.address_1,20,30,7,DF_RETURN | DF_ESC | DF_UP | DF_DOWN);
    switch (dir)
    {
        case DIR_ESC:
            goto END;
        case DIR_UP:
            goto FULL_NAME;
    }

ADDRESS_2:
    editstr (customer.address_2,20,30,8,DF_RETURN | DF_ESC | DF_UP | DF_DOWN);
    switch (dir)
    {
        case DIR_ESC:
            goto END;
        case DIR_UP:
            goto ADDRESS_1;
    }

ADDRESS_3:
    editstr (customer.address_3,20,30,9,DF_RETURN | DF_ESC | DF_UP);
    switch (dir)
    {
        case DIR_UP:
            goto ADDRESS_2;
    }

END:
    Write (customerfile,customerptr,&customer,sizeof customer);
}

void add_customer (void)
{
    printstr (10,5,"Account Name",12);
    customer = blankcustomer;
    createindex (customerfile,&customerptr,&customer,sizeof
customer,customermiscoff,customer.account_name,20,customerindexfile,30,5,DF_RETURN |
DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    alter_customer ();
    return;
}

```

```

CANCEL:
    deleteindex
(customerfile,customerptr,&customer,customermiscoff,customer.account_name,20,customerindexfile);
}

```

```

void edit_customer (void)
{
    printstr (10,5,"Account Name",12);
    select (customerfile,customerindexfile,&customerptr,&customer,sizeof
customer,customer.account_name,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    alter_customer ();
}

```

```

CANCEL;;
}

```

```

void display_customer (void)
{
    printstr (10,5,"Account Name",12);
    select (customerfile,customerindexfile,&customerptr,&customer,sizeof
customer,customer.account_name,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    show_customer ();
    wait ();
}

```

```

CANCEL;;
}

```

```

void remove_customer (void)
{
    char sure[1];

    printstr (10,5,"Account Name",12);
    select (customerfile,customerindexfile,&customerptr,&customer,sizeof
customer,customer.account_name,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:

```

```

        goto CANCEL;
    }
    show_customer ();
    printstr (30,20,"Are you sure? (Y/N) ",20);
    cursor (50,20);
    sure[0] = getkey ();
    if (!memicmp (sure,"Y",1))
        deleteindex
(customerfile,customerptr,&customer,customermiscoff,customer.account_name,20,customerinde
xfile);

```

```

CANCEL::
}

```

```

void enter_transactions_show (void)
{
    printstr (0,(-1),transaction.title,20);
    printdate (30,(-1),transaction.transaction_date);
    printstr (50,(-1),transaction.type,1);
}

```

```

void enter_transactions_edit (void)
{
    switch (fieldno)
    {
        case 0:
            goto TITLE;
        case 1:
            goto LOAN_DATE;
        case 2:
            goto TYPE;
    }
}

```

```

TITLE:
    editstr (transaction.title,20,0,(-1),DF_RETURN | DF_ESC | DF_UP | DF_DOWN |
DF_RIGHT | DF_PGUP | DF_PGDN | DF_HOME | DF_END | DF_INS | DF_DEL);
    switch (dir)
    {
    }
    return;

```

```

LOAN_DATE:
    editdate (&transaction.transaction_date,30,(-1),DF_RETURN | DF_ESC | DF_UP |
DF_DOWN | DF_LEFT | DF_RIGHT | DF_PGUP | DF_PGDN | DF_HOME | DF_END |
DF_INS | DF_DEL);
    switch (dir)
    {

```

```

    }
    return;

TYPE:
    editstr (transaction.type,1,50,(-1),DF_RETURN | DF_ESC | DF_UP | DF_DOWN | DF_LEFT |
DF_PGUP | DF_PGDN | DF_HOME | DF_END | DF_INS | DF_DEL);
    switch (dir)
    {
    }
    return;
}

void enter_transactions (void)
{
    printstr (10,5,"Account Name",12);
    select (customerfile,customerindexfile,&customerptr,&customer,sizeof
customer,customer.account_name,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    show_customer ();
    printstr (0,11,"Film Title",10);
    printstr (30,11,"Transaction Date",16);
    printstr (50,11,"Type (L/R)",10);
    scrolledit (transactionfile,&transactionptr,&transaction,foffset (transaction,customerptr),sizeof
transaction,customerfile,customerptr,&customer,foffset (customer,transactionfirst),sizeof
customer,&blankt

CANCEL::
}

void display_transactions (void)
{
    printstr (10,5,"Account Name",12);
    select (customerfile,customerindexfile,&customerptr,&customer,sizeof
customer,customer.account_name,20,30,5,DF_RETURN | DF_ESC);
    switch (dir)
    {
        case DIR_ESC:
            goto CANCEL;
    }
    show_customer ();
    printstr (0,11,"Film Title",10);
    printstr (30,11,"Transaction Date",16);
    printstr (50,11,"Type (L/R)",10);

```



```
    scrolledit (transactionfile,&transactionptr,&transaction,foffset (transaction,customerptr),sizeof
transaction,customerfile,customerptr,&customer,foffset (customer,transactionfirst),sizeof
customer,&blankt
```

```
CANCEL;;
}
```

```
execmnuitem menu00[] =
{
    "Add a Video",0,add_video,
    "Edit a Video",0,edit_video,
    "Display a Video",0,display_video,
    "Remove a Video",0,remove_video,
    0
};
```

```
execmnuitem menu01[] =
{
    "Add a Customer",0,add_customer,
    "Edit a Customer",0,edit_customer,
    "Display a Customer",0,display_customer,
    "Remove a Customer",0,remove_customer,
    0
};
```

```
execmnuitem menu02[] =
{
    "Enter Transactions",0,enter_transactions,
    "Display Transactions",0,display_transactions,
    0
};
```

```
execmnuitem menu[] =
{
    "Videos",menu00,0,
    "Customers",menu01,0,
    "Transactions",menu02,0,
    0
};
```

```
main (void)
{
    long r;
```

```
    transactionfile = opencreate ("transaction.dat");
    customerfile = opencreate ("customer.dat");
    customerindexfile = opencreate ("customer.idx");
```

```

videofile = opencreate ("video.dat");
videoindexfile = opencreate ("video.idx");

miscfile = open ("misc.dat",O_RDWR);
if (miscfile < 0)
{
    miscfile = open ("misc.dat",O_CREAT | O_TRUNC | O_RDWR,0777);
    if (miscfile < 0)
    {
        printf ("Can't create file misc.dat\n");
        return 1;
    }
    mblank3 ();
    mblank4 ();
    mblank4 ();
}

initvideo ();
execmenu (menu,-1,-1,0,0);
return 0;
}

```

## Appendix D DPG Source Code

### D.1 VIDEO.H

```

#define BLANKDATE      (1 + 1*32 + 80*32*16)

#define dateday(d)     ((d) % 32)
#define datemonth(d)   (((d) / 32) % 16)
#define dateyear(d)    ((d) / (32*16))

#define makedate(d,m,y) (y*32*16 + m*32 + d)

#define foffset(s,f)   ((char *)&s.f - (char *)&s)

enum
{
    DIR_RETURN,
    DIR_ESC,
    DIR_UP,
    DIR_DOWN,
    DIR_LEFT,
    DIR_RIGHT,

```

```

DIR_PGUP,
DIR_PGDN,
DIR_HOME,
DIR_END,
DIR_INS,
DIR_DEL,
MAXDIR
};

enum
{
    DF_RETURN = 1 << DIR_RETURN,
    DF_ESC = 1 << DIR_ESC,
    DF_UP = 1 << DIR_UP,
    DF_DOWN = 1 << DIR_DOWN,
    DF_LEFT = 1 << DIR_LEFT,
    DF_RIGHT = 1 << DIR_RIGHT,
    DF_PGUP = 1 << DIR_PGUP,
    DF_PGDN = 1 << DIR_PGDN,
    DF_HOME = 1 << DIR_HOME,
    DF_END = 1 << DIR_END,
    DF_INS = 1 << DIR_INS,
    DF_DEL = 1 << DIR_DEL,
};

typedef struct exccmenuitemstruct
{
    char *text;
    struct exccmenuitemstruct *submenu;
    void (*function)();
} exccmenuitem;

extern dir;

void printint (int,int,long,int);
void printfloat (int,int,double,int,int);
void printdate (int,int,unsigned short);
void printstr (int,int,char *,int);
void editlong (long *,int,int,int,int);
void editbyte (signed char *,int,int,int,int);
void editshort (short *,int,int,int,int);
void editfloat (double *,int,int,int,int,int);
void editdate (unsigned short *,int,int,int);
void editstr (char *,int,int,int,int);
void exccmenu (exccmenuitem *,int,int,int,int);
getkey (void);
void cursor (int,int);

```

```
validdate (unsigned short);
void beep (void);
unsigned short atodate (char *);
```

## D.2 VIDEO.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include "video.h"
#include "files.h"

#define background 0 /* Background colour */
#define foreground 7 /* Foreground colour */

#define attr      (((background << 4) | foreground) << 8)

#define seg(p)    *((unsigned *)&(p) + 1)
#define off(p)    *((unsigned *)&(p))

static char datemaxday[] =
{
    0,
    31,
    28,
    31,
    30,
    31,
    30,
    31,
    31,
    30,
    31,
    30,
    31,
    30,
    31,
};

int dir; /* Cursor direction */
int fieldno; /* Current field number for scroll edit */
static videopage; /* Segment part of pointer to video memory */
static currenty; /* Current Y coordinate for scrolling screens */

/* Check if last keystroke is an allowed cursor key, and if so, set
dir variable accordingly */
```

```

static setdir (int k,int directions)
{
    dir = -1;
    switch (k)
    {
        case '\r':
            if (directions & DF_RETURN)
                dir = DIR_RETURN;
            break;
        case 27:
            if (directions & DF_ESC)
                dir = DIR_ESC;
            break;
        case 18432:
            if (directions & DF_UP)
                dir = DIR_UP;
            break;
        case 20480:
            if (directions & DF_DOWN)
                dir = DIR_DOWN;
            break;
        case 19200:
            if (directions & DF_LEFT)
                dir = DIR_LEFT;
            break;
        case 19712:
            if (directions & DF_RIGHT)
                dir = DIR_RIGHT;
            break;
        case 18688:
            if (directions & DF_PGUP)
                dir = DIR_PGUP;
            break;
        case 20736:
            if (directions & DF_PGDN)
                dir = DIR_PGDN;
            break;
        case 18176:
            if (directions & DF_HOME)
                dir = DIR_HOME;
            break;
        case 20224:
            if (directions & DF_END)
                dir = DIR_END;
            break;
        case 20992:

```

```

        if (directions & DF_INS)
            dir = DIR_INS;
        break;
    case 21248:
        if (directions & DF_DEL)
            dir = DIR_DEL;
        break;
    }
    return dir >= 0;
}

/* Return pointer into video memory for given X,Y coordinates */

static int far *calcptr (int x,int y)
{
    int far *p;

    if (y < 0)
        y = currenty;
    currenty = y;
    seg (p) = videopage;
    off (p) = y*2*80 + x*2;
    return p;
}

/* Display a null-terminated ASCII string */

static void printzstr (int x,int y,char *s)
{
    int far *p;

    p = calcptr (x,y);
    while (*s)
        *p++ = *s++ + attr;
}

/* Convert an integer to ASCII representation */

static void zsprintint (char *s,long n,int l)
{
    char buf[20];

    sprintf (buf,"%%%d",l);
    sprintf (s,buf,n);
}

/* Convert a floating point number to ASCII representation */

```

```

static void zsprintfloat (char *s,double n,int l,int dec)
{
    char buf[20];

    if (dec <= 0)
        sprintf (buf,"%%%d.0lf",l);
    else
        sprintf (buf,"%%%d.%dlf",l + dec + 1,dec);
    sprintf (s,buf,n);
}

    /* Convert a date to ASCII representation */

static void zsprintdate (char *s,unsigned short d)
{
    sprintf (s,"%02d.%02d.%02d",dateday (d),datemonth (d),dateyear (d));
}

    /* Display a character a given number of times */

static void printchars (int x,int y,int n,int c)
{
    int far *p;

    p = calcptr (x,y);
    while (--n >= 0)
        *p++ = c + attr;
}

    /* Display a character */

static void printchar (int x,int y,int c)
{
    printchars (x,y,1,c);
}

    /* Draw an outline border using IBM graphics characters */

static void border (int x,int y,int width,int height)
{
    int i;

    printchar (x,y,0xda);
    printchars (x + 1,y,width - 2,0xc4);
    printchar (x + width - 1,y,0xbf);
    for (i=height-2; i; i--)

```

```

{
    putchar (x,y + i,0xb3);
    putchar (x + width - 1,y + i,0xb3);
}
    putchar (x,y + height - 1,0xc0);
    printf (x + 1,y + height - 1,width - 2,0xc4);
    putchar (x + width - 1,y + height - 1,0xd9);
}

```

/\* Allocate memory with error checking \*/

```

static void *alloc (int n)
{
    void *p;

    p = malloc (n);
    if (p == 0)
    {
        printf ("Out of memory");
        exit (1);
    }
    return p;
}

```

/\* Save a portion of the screen in a buffer allocated for the purpose \*/

```

static void *stashvideo (int x,int y,int width,int height)
{
    int *result,*dest;
    int far *src, far *q;
    int i;

    src = calcptr (x,y);
    result = dest = alloc (width*height*2);
    do
    {
        q = src;
        i = width;
        do
            *dest++ = *q++;
        while (--i);
        src += 80;
    }
    while (--height);
    return result;
}

```



```
/* Restore a portion of the screen from a buffer and free the buffer */
```

```
static void fetchvideo (int x,int y,int width,int height,void *data)
{
    int far *dest, far *p;
    int *src;
    int i;

    dest = calcptr (x,y);
    src = data;
    do
    {
        p = dest;
        i = width;
        do
            *p++ = *src++;
        while (--i);
        dest += 80;
    }
    while (--height);
    free (data);
}
```

```
void beep (void)
{
    putchar (7);
    fflush (stdout);
}
```

```
/* Initialize variables and clear the screen */
```

```
void initvideo (void)
{
    union REGS r;
    int far *p, far *q;
    int i,j;

    r.x.ax = 0x0f00;
    int86 (0x10,&r,&r);
    videopage = 0xb800;
    if (r.h.al == 7)
        videopage = 0xb000;
    seg (p) = videopage;
    off (p) = 0;
    i = 25;
    do
    {
```

```

    q = p;
    p += 80;
    j = 80;
    do
        *q++ = attr;
        while (--j != 0);
    }
    while (--i != 0);
}

/* Put hardware cursor at given coordinates */

void cursor (int x,int y)
{
    union REGS r;

    if (y < 0)
        y = currenty;
    r.h.ah = 2;
    r.h.dh = y;
    r.h.dl = x;
    r.h.bh = 0;
    int86 (0x10,&r,&r);
}

/* Get a keystroke from the user */

getkey (void)
{
    int k;

    k = bioskey (0);
    if (k & 0xff)
        k &= 0xff;
    return k;
}

/* Display a string with given number of characters */

void printstr (int x,int y,char *s,int l)
{
    int far *p;

    p = calcptr (x,y);
    while (--l >= 0)
        *p++ = *s++ + attr;
}

```

```
/* Display an integer using given number of digits */
```

```
void printint (int x,int y,long n,int l)
{
    char s[20];
    zsprintf (s,n,l);
    printzstr (x,y,s);
}
```

```
/* Display a floating point number using given number of digits */
```

```
void printfloat (int x,int y,double n,int l,int dec)
{
    char s[20];

    zsprintfloat (s,n,l,dec);
    printzstr (x,y,s);
}
```

```
/* Display a date */
```

```
void printdate (int x,int y,unsigned short d)
{
    char s[20];

    zsprintfdate (s,d);
    printzstr (x,y,s);
}
```

```
/* Input integer value from user */
```

```
void editlong (long *n,int l,int x,int y,int directions)
{
    int _l;
    int k;
    int changed;
    char s[20];

    cursor (x,y);
LOOP:
    changed = 0;
    zsprintf (s,*n,l);
    _l = 0;
    for (;;)
    {
        printzstr (x,y,s);
```

```

k = getkey ();
if (setdir (k,directions))
{
    *n = atol (s);
    printint (x,y,*n,l);
    return;
}
if (k == 8)
{
    if (_l > 0)
    {
        memmove (s + 1,s,l - 1);
        s[0] = ' ';
        _l--;
        continue;
    }
    goto LOOP;
}
if (k >= '0' && k <= '9' && _l != 1)
{
    if (!changed)
    {
        changed = 1;
        strset (s, ' ');
    }
    memmove (s,s + 1,l - 1);
    s[l - 1] = k;
    _l++;
}
}

void editbyte (signed char *n,int l,int x,int y,int directions)
{
    long t;

    t = *n;
    editlong (&t,l,x,y,directions);
    *n = t;
}

void editshort (short *n,int l,int x,int y,int directions)
{
    long t;

    t = *n;
    editlong (&t,l,x,y,directions);
}

```

```

    *n = t;
}

    /* Input floating point value from user */

void editfloat (double *n,int l,int dec,int x,int y,int directions)
{
    int _l,_dec;
    int k;
    int changed;
    char s[20];

    if (dec == 0)
        dec = -1;
    cursor (x,y);
LOOP:
    changed = 0;
    zsprintfloat (s,*n,l,dec);
    _l = 0;
    _dec = -1;
    for (;;)
    {
        printzstr (x,y,s);
        k = getkey ();
        if (setdir (k,directions))
        {
            *n = atof (s);
            printfloat (x,y,*n,l,dec);
            return;
        }
        if (k == 8)
        {
            if (_dec > 0)
            {
                s[l + _dec] = ' ';
                _dec--;
                continue;
            }
            if (_dec == 0)
            {
                _dec = -1;
                continue;
            }
            if (_l > 0)
            {
                memmove (s + 1,s,l - 1);
                s[0] = ' ';
            }
        }
    }
}

```

```

        _l--;
        continue;
    }
    goto LOOP;
}
if (k >= '0' && k <= '9')
{
    if (!changed)
    {
        changed = 1;
        strset (s, ' ');
        if (dec >= 0)
            s[l] = '.';
    }
    if (_dec < 0)
    {
        if (_l == 1)
            continue;
        memmove (s,s + 1,l - 1);
        s[l - 1] = k;
        _l++;
    }
    else
    {
        if (_dec == dec)
            continue;
        _dec++;
        s[l + _dec] = k;
    }
}
if (k == '.' && _dec < 0 && dec > 0)
{
    if (!changed)
    {
        zsprintfloat (s,0,l,dec);
        changed = 1;
    }
    _dec = 0;
}
}
}

```

/\* Create a date from ASCII representation \*/

```

unsigned short atodate (char *s)
{
    int day,month,year;

```

```

    day = atoi (s);
    month = atoi (s + 3);
    year = atoi (s + 6);
    return makedate (day,month,year);
}

```

```

    /* Check whether a date is valid */

```

```

validdate (unsigned short d)
{
    int day,month,year;

    day = dateday (d);
    month = datemonth (d);
    year = dateyear (d);

    if (month < 1 || month > 12)
        return 0;

    datemaxday[2] = 28;
    if ((year % 4) == 0 && year != 0)
        datemaxday[2] = 29;
    if (day < 1 || day > datemaxday[month])
        return 0;

    return 1;
}

```

```

    /* Input date from user */

```

```

void editdate (unsigned short *d,int x,int y,int directions)
{
    int k,i;
    int changed;
    char s[20];

    cursor (x,y);
LOOP:
    i = 0;
    changed = 0;
    zsprintdate (s,*d);
    for (;;)
    {
        printzstr (x,y,s);
        k = getkey ();
        if (setdir (k,directions))

```

```

{
  if (!validdate (atodate (s)))
  {
    beep ();
    goto LOOP;
  }
  *d = atodate (s);
  printdate (x,y,*d);
  return;
}
if (k == 8)
{
  if (i == 0)
    goto LOOP;
  if (i == 3 || i == 6)
    i--;
  s[--i] = ' ';
}
if (k >= '0' && k <= '9' && i != 8)
{
  if (!changed)
  {
    strcpy (s, " . . ");
    changed = 1;
  }
  s[i++] = k;
  if (i == 2 || i == 5)
    i++;
}
if (k == '.')
  switch (i)
  {
    case 0:
      if (!changed)
      {
        strcpy (s, " . . ");
        changed = 1;
      }
      memcpy (s,"01",2);
      i = 3;
      break;
    case 1:
      s[1] = s[0];
      s[0] = '0';
      i = 3;
      break;
    case 3:

```



```

        memcpy (s + 3,"01",2);
        i = 6;
        break;
    case 4:
        s[4] = s[3];
        s[3] = '0';
        i = 6;
        break;
    }
}
}

/* Input string from user */

void editstr (char *s,int l,int x,int y,int directions)
{
    char olds[256];
    int i;
    int k;
    int changed;

    cursor (x,y);
    memcpy (olds,s,l);
    i = 0;
    changed = 0;
    for (;;)
    {
        k = getkey ();
        if (setdir (k,directions))
        {
            if (changed)
                printchars (x,y,i - 1, ' ');
            return;
        }
        if (k >= 0x20 && k <= 0x7e && i != l)
        {
            if (!changed)
            {
                changed = 1;
                printchars (x,y,l, ' ');
                memset (s, ' ',l);
            }
            printchar (x + i,y,k);
            s[i++] = k;
        }
        if (k == 8)
        {

```

```

    if (i)
    {
        s[--i] = ' ';
        printchar (x + i,y, ' ');
    }
    else
    {
        memcpy (s,olds,l);
        printstr (x,y,s,l);
        changed = 0;
    }
}
}
}

```

```

static void printexecmenuitem (int x,int y,int i,execmenuitem *m,int maxwidth,
    int reverse)

```

```

{
    int far *p;
    int a;
    char *s;

    p = calcptr (x + 1,y + i + 1);
    a = attr;
    if (reverse)
        a = ((foreground << 4) | background) << 8;
    *p++ = ' ' + a;
    s = m->text;
    while (*s)
    {
        *p++ = *s++ + a;
        maxwidth--;
    }
    while (maxwidth-- >= 0)
        *p++ = ' ' + a;
}

```

/\* Display menu on screen, allow user to select an item, and take appropriate action: if the selected item points to a function, call the function, otherwise display the submenu \*/

```

void execmenu (execmenuitem *m,int mx,int my,int prevwidth,int prevheight)
{
    int nitems;
    int maxwidth;
    int i,j,k,x,y;
    execmenuitem *n,*o;

```

```

void *oldvideo,*menuscreen;
char *s;

cursor (0,25);
nitems = 0;
maxwidth = 0;
for (n=m; n->text; n++)
{
    nitems++;
    if (strlen (n->text) > maxwidth)
        maxwidth = strlen (n->text);
}
x = 10;
y = 5;
if (mx == -2)
{
    x = 38 - maxwidth/2;
    y = 12 - nitems/2;
}
if (mx >= 0)
{
    x = mx + 8;
    if (x < mx + prevwidth - maxwidth - 3)
        x = mx + prevwidth - maxwidth - 3;
    if (x > mx + prevwidth - 1)
        x = mx + prevwidth - 1;
    y = my + 4;
    if (y < my + prevheight - nitems - 1)
        y = my + prevheight - nitems - 1;
    if (y > my + prevheight - 1)
        y = my + prevheight - 1;
}
if (x > 76 - maxwidth)
    x = 76 - maxwidth;
if (y > 23 - nitems)
    y = 23 - nitems;
oldvideo = stashvideo (x,y,maxwidth + 4,nitems + 2);
border (x,y,maxwidth + 4,nitems + 2);
n = m;
for (i=0; i!=nitems; i++)
{
    printzstr (x + 2,y + 1 + i,n->text);
    n++;
}
n = m;
i = 0;
for (;;)

```

```

{
  printexecmenuitem (x,y,i,n,maxwidth,1);
  k = getkey ();
  setdir (k,~0);
  switch (dir)
  {
    case DIR_ESC:
      fetchvideo (x,y,maxwidth + 4,nitems + 2,oldvideo);
      return;
    case DIR_UP:
      if (i)
      {
        printexecmenuitem (x,y,i,n,maxwidth,0);
        i--;
        n--;
      }
      break;
    case DIR_DOWN:
      if (i != nitems - 1)
      {
        printexecmenuitem (x,y,i,n,maxwidth,0);
        i++;
        n++;
      }
      break;
    case DIR_PGUP:
    case DIR_HOME:
      if (i)
        printexecmenuitem (x,y,i,n,maxwidth,0);
      i = 0;
      n = m;
      break;
    case DIR_PGDN:
    case DIR_END:
      if (i != nitems - 1)
        printexecmenuitem (x,y,i,n,maxwidth,0);
      i = nitems - 1;
      n = m + i;
      break;
    case DIR_RETURN:
      o = n;
      goto DO_OPTION;
  }
  if (k == '0')
  {
    j = 9;
  }
  SELECT_NUMBER:

```

```

    if (j >= nitems)
        continue;
    o = m + j;
    goto DO_OPTION_1;
}
if (k >= '1' && k <= '9')
{
    j = k - '1';
    goto SELECT_NUMBER;
}
if (k >= 0x3b00 && k <= 0x4400)
{
    j = (k - 0x3b00) / 0x100;
    goto SELECT_NUMBER;
}
k = toupper (k);
j = 0;
for (o=m; o->text; o++)
{
    s = o->text;
    while (*s >= 'a' && *s <= 'z')
        s++;
    if (*s == k)
    {
DO_OPTION_1:
        printexecmenuitem (x,y,i,n,maxwidth,0);
        i = j;
        n = o;
        printexecmenuitem (x,y,i,n,maxwidth,1);
DO_OPTION:
        if (o->submenu)
            execmenu (o->submenu,x,y,maxwidth + 4,nitems + 2);
        else
        {
            menuseen = stashvideo (0,0,80,25);
            printchars (0,0,80*25, ' ');
            (o->function)();
            fetchvideo (0,0,80,25,menuseen);
            cursor (0,25);
        }
        break;
    }
    j++;
}
}
}

```

```

static void scrollup (int x,int y,int width,int height)
{
    int far *scr, far *p, far *q;
    int i,j;

    scr = calcptr (x,y);
    j = height;
    while (--j)
    {
        p = scr;
        q = scr + 80;
        i = width;
        do
            *p++ = *q++;
        while (--i);
        scr += 80;
    }
    if (height)
        printchars (x,y + height - 1,width,' ');
}

```

```

static void scrolldown (int x,int y,int width,int height)
{
    int far *scr, far *p, far *q;
    int i,j;

    scr = calcptr (x,y + height);
    j = height;
    while (--j)
    {
        scr -= 80;
        p = scr;
        q = scr - 80;
        i = width;
        do
            *p++ = *q++;
        while (--i);
    }
    if (height)
        printchars (x,y,width,' ');
}

```

/\* Function to display linked list of records in scrolling format, using supplied functions to show and (optionally) edit records \*/

```

void scrolledit (
    int fromfile, long *fromptr, void *fromrec, int fromoffset, int fromsize,

```

```

    int tofile,long toptr,void *torec,int toffset,int tosize,
    void *blankrec,int miscoff,int nfields,void (*show)(),void (*edit)(),
    int x,int y,int width,int height)
{
    long *fromptrs;
    long *toptrs;
    long p,q;
    int n;
    long pv[25];
    int i,j;

    fromptrs = (long *)(((char *)fromrec) + fromoffset);
    toptrs = (long *)(((char *)torec) + toffset);
    p = toptrs[0];
    if (p < 0)
    {
        if (!edit)
        {
            printzstr (30,23,"No records to display");
            cursor (53,23);
            beep ();
            getkey ();
            return;
        }
        memcpy (fromrec,blankrec,fromsize);
        create (fromfile,fromptr,fromrec,fromsize,miscoff);
        remlink (fromfile,*fromptr,fromptrs,fromoffset,tofile,toptr,toptrs,toffset);
        p = toptrs[0];
    }
    n = 0;
    do
    {
        *fromptr = pv[n] = p;
        Read (fromfile,p,fromrec,fromsize);
        currenty = y + n;
        show ();
        p = fromptrs[1];
        n++;
    }
    while (n != height && p >= 0);
    *fromptr = pv[0];
    Read (fromfile,pv[0],fromrec,fromsize);
    fieldno = 0;
    i = 0;
    for (;;)
    {
        currenty = y + i;

```

```

if (edit)
    edit ();
else
{
    cursor (x,currenty);
    do
        i = getkey ();
    while (!setdir (i,DF_ESC | DF_UP | DF_DOWN | DF_PGUP | DF_PGDN | DF_HOME |
DF_END));
}
switch (dir)
{
case DIR_RETURN:
    if (fieldno == nfields - 1)
    {
        Write (fromfile,*fromptr,fromrec,fromsize);
        fieldno = 0;
        if (fromptrs[1] >= 0)
            goto DOWN;
        memcpy (fromrec,blankrec,fromsize);
        create (fromfile,fromptr,fromrec,fromsize,miscoeff);
        toptrs[1] = *fromptr;
        fromptrs[0] = toptr;
        fromptrs[2] = pv[i];
        Write (fromfile,pv[i] + fromoffset + sizeof (long),fromptr,sizeof (long));
        if (n != height - 1)
        {
            n++;
            i++;
            currenty++;
        }
    }
    else
    {
        memmove (pv,pv + 1,sizeof pv - sizeof (long));
        scrollup (x,y,width,height);
    }
    pv[i] = *fromptr;
    show ();
}
else
    fieldno++;
break;
case DIR_ESC:
    if (edit)
    {
        Write (fromfile,*fromptr,fromrec,fromsize);
        Write (tofile,toptr,torec,tosize);
    }
}

```



```

    }
    return;
case DIR_UP:
    if (i)
    {
        if (edit)
            Write (fromfile,*fromptr,fromrec,fromsize);
        i--;
        *fromptr = pv[i];
        Read (fromfile,*fromptr,fromrec,fromsize);
    }
    else
        if (fromptrs[2] >= 0)
        {
            if (edit)
                Write (fromfile,*fromptr,fromrec,fromsize);
            memmove (pv + 1,pv,sizeof pv - sizeof (long));
            *fromptr = pv[0] = fromptrs[2];
            Read (fromfile,pv[0],fromrec,fromsize);
            scrolldown (x,y,width,height);
            show ();
        }
        break;
case DIR_DOWN:
DOWN:
    if (i != n - 1)
    {
        if (edit)
            Write (fromfile,*fromptr,fromrec,fromsize);
        i++;
        *fromptr = pv[i];
        Read (fromfile,*fromptr,fromrec,fromsize);
    }
    else
        if (fromptrs[1] >= 0)
        {
            if (edit)
                Write (fromfile,*fromptr,fromrec,fromsize);
            memmove (pv,pv + 1,sizeof pv - sizeof (long));
            *fromptr = pv[i] = fromptrs[1];
            Read (fromfile,*fromptr,fromrec,fromsize);
            scrollup (x,y,width,height);
            show ();
        }
        break;
case DIR_LEFT:
    fieldno--;

```

```

    break;
case DIR_RIGHT:
    fieldno++;
    break;
case DIR_PGUP:
    if (edit)
        Write (fromfile,*fromptr,fromrec,fromsize);
    if (i)
    {
        *fromptr = pv[0];
        Read (fromfile,pv[0],fromrec,fromsize);
        currenty = y;
    }
    for (j=0; j!=height && fromptrs[2] >= 0; j++)
    {
        memmove (pv + 1,pv,sizeof pv - sizeof (long));
        *fromptr = pv[0] = fromptrs[2];
        Read (fromfile,pv[0],fromrec,fromsize);
        scrolldown (x,y,width,height);
        show ();
    }
    if (i)
    {
        *fromptr = pv[i];
        Read (fromfile,*fromptr,fromrec,fromsize);
    }
    break;
case DIR_PGDN:
    if (edit)
        Write (fromfile,*fromptr,fromrec,fromsize);
    if (i != n - 1)
    {
        *fromptr = pv[n - 1];
        Read (fromfile,*fromptr,fromrec,fromsize);
        currenty = y + n - 1;
    }
    for (j=0; j!=height && fromptrs[1] >= 0; j++)
    {
        memmove (pv,pv + 1,sizeof pv - sizeof (long));
        *fromptr = pv[n - 1] = fromptrs[1];
        Read (fromfile,*fromptr,fromrec,fromsize);
        scrollup (x,y,width,height);
        show ();
    }
    if (i != n - 1)
    {
        *fromptr = pv[i];

```

```

    Read (fromfile,*fromptr,fromrec,fromsize);
}
break;
case DIR_HOME:
if (fromptrs[2] >= 0)
{
    if (edit)
        Write (fromfile,*fromptr,fromrec,fromsize);
    p = toptrs[0];
    n = 0;
    do
    {
        *fromptr = pv[n] = p;
        Read (fromfile,p,fromrec,fromsize);
        currenty = y + n;
        show ();
        p = fromptrs[1];
        n++;
    }
    while (n != height && p >= 0);
    *fromptr = pv[0];
    Read (fromfile,pv[0],fromrec,fromsize);
    i = 0;
}
break;
case DIR_END:
if (fromptrs[1] >= 0)
{
    if (edit)
        Write (fromfile,*fromptr,fromrec,fromsize);
    i = n - 1;
    *fromptr = pv[i];
    Read (fromfile,*fromptr,fromrec,fromsize);
    while (fromptrs[1] >= 0)
    {
        memmove (pv,pv + 1,sizeof pv - sizeof (long));
        *fromptr = pv[i] = fromptrs[1];
        Read (fromfile,*fromptr,fromrec,fromsize);
    }
    for (i=0; i!=n; i++)
    {
        *fromptr = pv[i];
        Read (fromfile,*fromptr,fromrec,fromsize);
        currenty = y + i;
        show ();
    }
    i = n - 1;
}

```

```

    }
    break;
case DIR_INS:
    Write (fromfile,*fromptr,fromrec,fromsize);
    p = fromptrs[2];
    memcpy (fromrec,blankrec,fromsize);
    create (fromfile,fromptr,fromrec,fromsize,misloff);
    fromptrs[0] = toptr;
    fromptrs[1] = pv[i];
    fromptrs[2] = p;
    if (p < 0)
        toptrs[0] = *fromptr;
    else
        Write (fromfile,p + fromoffset + sizeof (long),fromptr,sizeof (long));
    Write (fromfile,pv[i] + fromoffset + 2*sizeof (long),fromptr,sizeof (long));
    if (n != height - 1)
        n++;
    memmove (pv + i + 1,pv + i,(n - i - 1)*sizeof (long));
    pv[i] = *fromptr;
    scrolldown (x,currenty,width,height + y - currenty);
    show ();
    break;
case DIR_DEL:
    if (n != 1)
    {
        p = fromptrs[1];
        q = fromptrs[2];
        recunlink (fromfile,*fromptr,fromptrs,fromoffset,tofile,toptr,toptrs,tooffset);
        delete (fromfile,*fromptr,fromrec,misloff);
        scrollup (x,y + currenty,width,height + y - currenty);
        if (n == height && p >= 0)
        {
            *fromptr = pv[n - 1] = p;
            Read (fromfile,p,fromrec,fromsize);
            currenty = y + height - 1;
            show ();
        }
    }
    else
        if (i == n - 1)
        {
            if (i)
            {
                printchars (x,y + i,width, ' ');
                i--;
                n--;
            }
            else

```

```

        {
            *fromptr = pv[0] = q;
            Read (fromfile,q,fromrec,fromsize);
            printchars (x,y,width,' ');
            show ();
        }
    }
else
    n--;
}
break;
}
}
}

```

### D.3 FILES.H

extern miscfile;

```

void Read (int,long,void *,int);
void Write (int,long,void *,int);
void delete (int,long,void *,long);
void deleteindex (int,long,void *,long,char *,int,int);
void create (int,long *,void *,int,long);
void createindex (int,long *,void *,int,long,char *,int,int,int,int,int);
void find (int,int,long *,void *,int,int,char *);
void select (int,int,long *,void *,int,char *,int,int,int,int);
opencreate (char *);
void mwrite (void *,int);
void blank3 (void);
void blank4 (void);
void remlink (int,long,long *,int,int,long,long *,int);
void recunlink (int,long,long *,int,int,long,long *,int);

```

### D.4 FILES.C

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <string.h>
#include <assert.h>
#include "video.h"
#include "files.h"

```

```

#define MAXKEY    32  /* Maximum depth a tree can achieve */
#define NODESIZE  512 /* Physical size of each node */

                /* Macros to access values from nodes in memory */

#define nentries    (*(short*)(node + NODESIZE - sizeof(short)))
#define nodeptr(i)  ((long*)node)[i]
#define recptr(i)   ((long*)node)[i + m]
#define entry(i)    (node + (m*2 - 1)*sizeof(long) + (i)*length)

#define nentries2   (*(short*)(node2 + NODESIZE - sizeof(short)))
#define nodeptr2(i) ((long*)node2)[i]
#define recptr2(i)  ((long*)node2)[i + m]
#define entry2(i)   (node2 + (m*2 - 1)*sizeof(long) + (i)*length)

#define nentries3   (*(short*)(node3 + NODESIZE - sizeof(short)))
#define nodeptr3(i) ((long*)node3)[i]
#define recptr3(i)  ((long*)node3)[i + m]
#define entry3(i)   (node3 + (m*2 - 1)*sizeof(long) + (i)*length)

                /* Storage for nodes in memory */
static char node[NODESIZE];
static char node2[NODESIZE];
static char node3[NODESIZE];
static m;          /* Maximum branching factor */
static length;    /* Length of keys */
static indexfile; /* Handle for index file */
static miscoff;   /* Offset of data in miscfile */
static long newnodeptr; /* Pointer to newly created node */

int miscfile;

                /* Read data from file with error checking */

void Read (int file,long p,void *buf,int bytes)
{
    lseek (file,p,SEEK_SET);
    if (read (file,buf,bytes) != bytes)
    {
        printf ("Read Error\n");
        exit (1);
    }
}

                /* Write data to file with error checking */

```

```

void Write (int file,long p,void *buf,int bytes)
{
    lseek (file,p,SEEK_SET);
    if (write (file,buf,bytes) != bytes)
    {
        printf ("Write Error\n");
        exit (1);
    }
}

    /* Calculate the maximum branching factor given the key length for the
        current index file */

static void calcm (void)
{
    m = (NODESIZE + length + sizeof (long) - sizeof (short)) /
        (length + 2*sizeof (long));
}

    /* Allocate a new node */

static long allocnode (void)
{
    long p;
    long next;

    /* Pointer to first node on free list */

    Read (miscfile,miscoff + 3*sizeof (long),&p,sizeof p);

    if (p < 0) /* If no nodes on free list, add one to end of file */
        return filelength (indexfile);

    /* Return first node on free list and update list pointer */

    Read (indexfile,p,&next,sizeof next);
    Write (miscfile,miscoff + 3*sizeof (long),&next,sizeof next);
    return p;
}

    /* Add a node to free list */

static void freenode (long t)
{
    long p;

    Read (miscfile,miscoff + 3*sizeof (long),&p,sizeof p);

```

```

Write (indexfile,t,&p,sizeof p);
Write (miscfile,miscoeff + 3*sizeof (long),&t,sizeof t);
}

/* Delete a record from linked list of records, and add it to list of
   free data records */

void delete (int file,long ptr,void *rec,long _miscoeff)
{
    long listptr[3];
    long next;
    long prev;

    miscoeff = _miscoeff;

    next = ((long *)rec)[0]; /* Pointer to next record */
    prev = ((long *)rec)[1]; /* Pointer to previous record */

    /* Remove record from linked list */

    if (next >= 0)
        Write (file,next + sizeof (long),&prev,sizeof prev);
    if (prev >= 0)
        Write (file,prev,&next,sizeof next);
    Read (miscfile,miscoeff,listptr,sizeof listptr);
    if (listptr[1] == ptr)
        listptr[1] = next;
    if (listptr[2] == ptr)
        listptr[2] = prev;

    /* Add record to free list */

    Write (file,ptr,&listptr[0],sizeof listptr[0]);
    listptr[0] = ptr;

    Write (miscfile,miscoeff,listptr,sizeof listptr);
}

/* Delete a data record from an indexed file */

void deleteindex (int file,long ptr,void *rec,long _miscoeff,
                 char *key,int _length,int _indexfile)
{
    long t,t2;
    int n;
    int lo,hi;
    int i,j,k;

```



```

int c;
int iv[MAXKEY];
long chain[MAXKEY];

indexfile = _indexfile;
miscoff = _miscoff;
length = _length;
calcm ();

    /* Delete record from linked list */

delete (file,ptr,rec,miscoff);

t = 0;
n = -1;
for (;;)
{
    Read (indexfile,t,node,NODESIZE);
    lo = 0;
    hi = nentries - 1;
    while (lo <= hi)
    {
        i = (lo + hi) / 2;
        c = memicmp (entry (i),key,length);
        if (c == 0)
            break;
        if (c < 0)
            lo = i + 1;
        else
            hi = i - 1;
    }
    if (c == 0)
        break;
    n++;
    assert (n < MAXKEY);
    iv[n] = lo;
    chain[n] = t;
    t = nodeptr (lo);
    assert (t >= 0);
}
if (nodeptr (0) >= 0)
{
    t2 = t;
    lo++;
    n++;
    assert (n < MAXKEY);
    iv[n] = lo;
}

```

```

chain[n] = t;
t = nodeptr (lo);
assert (t >= 0);
Read (indexfile,t,node,NODESIZE);
do
{
    n++;
    assert (n < MAXKEY);
    iv[n] = 0;
    chain[n] = t;
    t = nodeptr (0);
    assert (t >= 0);
    Read (indexfile,t,node,NODESIZE);
}
while (nodeptr (0) >= 0);
Write (indexfile,t2 + (m+lo)*sizeof (long),&recptr (lo),sizeof (long));
Write (indexfile,t2 + m*2*sizeof (long) + lo*length,entry (lo),length);
}
memmove (&nodeptr (lo),&nodeptr (lo + 1),(nentries - lo)*sizeof (long));
memmove (&recptr (lo),&recptr (lo + 1),(nentries - lo - 1)*sizeof (long));
memmove (entry (lo),entry (lo + 1),(nentries - lo - 1)*length);
nentries--;
Write (indexfile,t,node,NODESIZE);

    /* While current node has too few entries */

while (nentries < (m - 1) / 2 && n > 0)
{
    /* Read father node */

    Read (indexfile,chain[n - 1],node3,NODESIZE);

    /* Index for father key */

    i = iv[n - 1] - 1;

    /* If current node is leftmost, merge with right brother */

    if (i < 0)
    {
        /* Read brother node */

        t2 = nodeptr3 (1);
        Read (indexfile,t2,node2,NODESIZE);

        /* If merged node would have few enough entries */

```

```

if (nentries + nentries2 < m - 1)
{
    /* Move over brother node keys */

    memmove (&nodeptr2 (nentries + 1),&nodeptr2 (0),(nentries2 + 1)*sizeof (long));
    memmove (&recptr2 (nentries + 1),&recptr2 (0),nentries2*sizeof (long));
    memmove (entry2 (nentries + 1),entry2 (0),nentries2*length);
    nentries2 += nentries + 1;

    /* Copy current node keys into brother node */

    memcpy (&nodeptr2 (0),&nodeptr (0),(nentries + 1)*sizeof (long));
    memcpy (&recptr2 (0),&recptr (0),nentries*sizeof (long));
    memcpy (entry2 (0),entry (0),nentries*length);

    /* Put father key into brother node */

    recptr2 (nentries) = recptr3 (0);
    memcpy (entry2 (nentries),entry3 (0),length);

    /* Delete father key from father node */

    memmove (&nodeptr3 (0),&nodeptr3 (1),nentries3*sizeof (long));
    memmove (&recptr3 (0),&recptr3 (1),(nentries3 - 1)*sizeof (long));
    memmove (entry3 (0),entry3 (1),(nentries3 - 1)*length);
    nentries3--;

    /* Free current node */

    freenode (t);

    /* If father node is root node and empty, delete it */

    if (n == 1 && nentries3 == 0)
    {
        /* Free prior location of brother node */

        freenode (t2);

        /* Write brother node into root node's location */

        Write (indexfile,0,node2,NODESIZE);
    }
    else /* Write back nodes */
    {
        /* Write back brother node */

```

```

Write (indexfile,t2,node2,NODESIZE);

    /* Write back father node */

Write (indexfile,chain[n - 1],node3,NODESIZE);
}

    /* Finish */

break;
}
else /* Redistribute keys between two nodes */
{
    /* Middle key to be used as replacement for father key */

j = (nentries2 + nentries) / 2 - nentries - 1;

    /* Copy in father key */

recptr (nentries) = recptr3 (0);
memcpy (entry (nentries),entry3 (0),length);

    /* Copy keys from brother node to current node */

memcpy (&nodeptr (nentries + 1),&nodeptr2 (0),(j + 1)*sizeof (long));
memcpy (&recptr (nentries + 1),&recptr2 (0),j*sizeof (long));
memcpy (entry (nentries + 1),entry2 (0),j*length);

    /* Copy middle key into father node */

recptr3 (0) = recptr2 (j);
memcpy (entry3 (0),entry2 (j),length);
nentries += j + 1;

    /* Move down entries in brother node */

memmove (&nodeptr2 (0),&nodeptr2 (j + 1),(nentries2 - j)*sizeof (long));
memmove (&recptr2 (0),&recptr2 (j + 1),(nentries2 - j - 1)*sizeof (long));
memmove (entry2 (0),entry2 (j + 1),(nentries2 - j - 1)*length);
nentries2 -= j + 1;

    /* Write back current node */

Write (indexfile,t,node,NODESIZE);

    /* Write back brother node */

```

```

Write (indexfile,t2,node2,NODESIZE);

    /* Write back father node */

Write (indexfile,chain[n - 1],node3,NODESIZE);

    /* Step up to father node and continue */

memcpy (node,node3,NODESIZE);
n--;
t = chain[n];
}
}
else /* Merge with left brother */
{
    /* Read brother node */

t2 = nodeptr3 (i);
Read (indexfile,t2,node2,NODESIZE);

    /* If merged node would have few enough entries */

if (nentries + nentries2 < m - 1)
{
    /* Put father key into brother node */

recptr2 (nentries2) = recptr3 (i);
memcpy (entry2 (nentries2),entry3 (i),length);
nentries2++;

    /* Merge current node keys into brother node */

memcpy (&nodeptr2 (nentries2),&nodeptr (0),(nentries + 1)*sizeof (long));
memcpy (&recptr2 (nentries2),&recptr (0),nentries*sizeof (long));
memcpy (entry2 (nentries2),entry (0),nentries*length);
nentries2 += nentries;

    /* Delete father key from father node */

memmove (&nodeptr3 (i + 1),&nodeptr3 (i + 2),(nentries3 - i)*sizeof (long));
memmove (&recptr3 (i),&recptr3 (i + 1),(nentries3 - i - 1)*sizeof (long));
memmove (entry3 (i),entry3 (i + 1),(nentries3 - i - 1)*length);
nentries3--;

    /* Free current node */

freenode (t);

```

```

        /* Write back brother node */

Write (indexfile,t2,node2,NODESIZE);

        /* Write back father node */

Write (indexfile,chain[n - 1],node3,NODESIZE);

        /* Finish */

break;
}
else /* Redistribute keys between two nodes */
{
        /* Middle key to be used as replacement for father key */

j = (nentries2 + nentries) / 2;

        /* Number of keys to transfer from brother to current node */

k = nentries2 - j;

        /* Move over keys in current node to make room */

memmove (&nodeptr (k + 1),&nodeptr (0),(nentries + 1)*sizeof (long));
memmove (&recptr (k + 1),&recptr (0),nentries*sizeof (long));
memmove (entry (k + 1),entry (0),nentries*length);
nentries += k + 1;

        /* Copy in father key */

recptr (k) = recptr3 (i);
memcpy (entry (k),entry3 (i),length);

        /* Copy keys from brother node to current node */

memcpy (&nodeptr (0),&nodeptr2 (j + 1),(k + 1)*sizeof (long));
memcpy (&recptr (0),&recptr2 (j + 1),k*sizeof (long));
memcpy (entry (0),entry2 (j + 1),k*length);
nentries2 -= k + 1;

        /* Copy middle key into father node */

recptr3 (i) = recptr2 (j);
memcpy (entry3 (i),entry2 (j),length);

```

```

        /* Write back current node */

Write (indexfile,t,node,NODESIZE);

        /* Write back brother node */

Write (indexfile,t2,node2,NODESIZE);

        /* Write back father node */

Write (indexfile,chain[n - 1],node3,NODESIZE);

        /* Step up to father node and continue */

        memcpy (node,node3,NODESIZE);
        n--;
        t = chain[n];
    }
}
}
}

/* Allocate a new data record, and add to linked list */

void create (int file,long *ptr,void *rec,int size,long _miscoff)
{
    long listptr[3];
    long p;

    miscoff = _miscoff;

    Read (miscfile,miscoff,listptr,sizeof listptr);

    if (listptr[0] < 0)      /* If free list is empty */
        p = filelength (file); /* get new record from end of file */
    else                    /* get first record from free list */
    {
        p = listptr[0];
        Read (file,p,listptr,sizeof (long));
    }

    if (listptr[2] < 0) /* If linked list is empty, initialize it */
    {
        listptr[1] = p;
        listptr[2] = p;
    }
    else                    /* add record to linked list */

```

```

{
    ((long *)rec)[1] = listptr[2];
    Write (file,listptr[2],&p,sizeof (long));
    listptr[2] = p;
}

Write (file,p,rec,size);
Write (miscfile,misccoff,listptr,sizeof listptr);
*ptr = p;
}

/* Recursively insert key in B-tree and return error if any */

static insert (long t,long *rptr,char *rkey,long ptr,char *key)
{
    long newptr;
    char newkey[MAXKEY];
    int lo,hi;
    int i,j;
    int c;
    long n1;

    if (t < 0) /* Run out of tree so have to back up */
    {
        *rptr = ptr;
        memcpy (rkey,key,length);
        return 0;
    }

    Read (indexfile,t,node,NODESIZE);

    /* Find where to insert new key in current node */

    lo = 0;
    hi = nentries - 1;
    while (lo <= hi)
    {
        i = (lo + hi) / 2;
        c = memicmp (entry (i),key,length);
        if (c == 0) /* Key already exists, so return error */
            return 1;
        if (c < 0)
            lo = i + 1;
        else
            hi = i - 1;
    }
}

```



```

    /* Recursively insert key below current node */

if (insert (nodeptr (i),&newptr,newkey,ptr,key))
    return 1;

    /* If insertion has propagated up from node below */

if (newptr >= 0)
{
    Read (indexfile,t,node,NODESIZE);

        /* Again find key in current node */

    lo = 0;
    hi = nentries - 1;
    while (lo <= hi)
    {
        i = (lo + hi) / 2;
        c = memicmp (entry (i),newkey,length);
        assert (c != 0);
        if (c < 0)
            lo = i + 1;
        else
            hi = i - 1;
    }
    if (nentries == m - 1)    /* If node is too full, split it */
    {
        n1 = allocnode ();
        j = m / 2;
        if (lo <= j)
            j--;
        *rptr = recptr (j);
        memcpy (rkey,entry (j),length);
        memmove (&nodeptr (j),&nodeptr (j + 1),(nentries - j)*sizeof (long));
        memmove (&recptr (j),&recptr (j + 1),(nentries - j - 1)*sizeof (long));
        memmove (entry (j),entry (j + 1),(nentries - j - 1)*length);
        if (lo <= j + 1)
            j++;
        memmove (&nodeptr (lo + 1),&nodeptr (lo),(nentries - lo)*sizeof (long));
        memmove (&recptr (lo + 1),&recptr (lo),(nentries - lo - 1)*sizeof (long));
        memmove (entry (lo + 1),entry (lo),(nentries - lo - 1)*length);
        nodeptr (lo) = newnodeptr;
        recptr (lo) = newptr;
        memcpy (entry (lo),newkey,length);
        nentries = j;
        Write (indexfile,n1,node,NODESIZE);
        nentries = m - 1 - j;
    }
}

```

```

    memcpy (&nodeptr (0),&nodeptr (j),(nentries + 1)*sizeof (long));
    memcpy (&recptr (0),&recptr (j),nentries*sizeof (long));
    memcpy (entry (0),entry (j),nentries*length);
    Write (indexfile,t,node,NODESIZE);
    newnodeptr = n1;
    return 0;
}
else          /* Simply insert in current node */
{
    memmove (&nodeptr (lo + 1),&nodeptr (lo),(nentries - lo + 1)*sizeof (long));
    memmove (&recptr (lo + 1),&recptr (lo),(nentries - lo)*sizeof (long));
    memmove (entry (lo + 1),entry (lo),(nentries - lo)*length);
    nodeptr (lo) = newnodeptr;
    recptr (lo) = newptr;
    memcpy (entry (lo),newkey,length);
    nentries++;
    Write (indexfile,t,node,NODESIZE);
}
}

*rptr = -1;
return 0;
}

```

/\* Allocate a new data record and insert index key in B-tree \*/

```

void createindex (int file,long *ptr,void *rec,int size,long _miscoff,
    char *key,int _length,int _indexfile,int x,int y,int directions)
{
    long listptr[3];
    long p;
    long newptr;
    char newkey[MAXKEY];
    int lo,hi;
    int i,j;
    int c;
    long n1,n2;
    long splitptr;
    char splitkey[MAXKEY];

    indexfile = _indexfile;
    miscoff = _miscoff;
    length = _length;
    calcm ();

    Read (miscfile,miscoff,listptr,sizeof listptr);
    if (listptr[0] < 0)      /* If free list is empty */

```

```

    p = filelength (file); /* get new record from end of file */
else /* get first record from free list */
{
    p = listptr[0];
    Read (file,p,listptr,sizeof (long));
}

```

REDO:

```

    /* Ask user for key value */

editstr (key,length,x,y,directions);

if (dir == DIR_ESC) /* User hit ESC, so cancel */
{
    *ptr = -1;
    return;
}

if (filelength (indexfile)) /* If index file not empty */
{
    newnodeptr = -1;
    if (insert (0,&newptr,newkey,p,key))
    {
        beep ();
        goto REDO;
    }
    if (newptr >= 0)
    {
        Read (indexfile,0,node,NODESIZE);
        lo = 0;
        hi = nentries - 1;
        while (lo <= hi)
        {
            i = (lo + hi) / 2;
            c = memicmp (entry (i),newkey,length);
            assert (c != 0);
            if (c < 0)
                lo = i + 1;
            else
                hi = i - 1;
        }
        if (nentries == m - 1)
        {
            n1 = allocnode ();
            n2 = allocnode ();
            j = m / 2;

```

```

if (lo <= j)
    j--;
splitptr = recptr (j);
memcpy (splitkey,entry (j),length);
memmove (&nodeptr (j),&nodeptr (j + 1),(nentries - j)*sizeof (long));
memmove (&recptr (j),&recptr (j + 1),(nentries - j - 1)*sizeof (long));
memmove (entry (j),entry (j + 1),(nentries - j - 1)*length);
if (lo <= j + 1)
    j++;
memmove (&nodeptr (lo + 1),&nodeptr (lo),(nentries - lo)*sizeof (long));
memmove (&recptr (lo + 1),&recptr (lo),(nentries - lo - 1)*sizeof (long));
memmove (entry (lo + 1),entry (lo),(nentries - lo - 1)*length);
nodeptr (lo) = newnodeptr;
recptr (lo) = newptr;
memcpy (entry (lo),newkey,length);
nentries = j;
Write (indexfile,n1,node,NODESIZE);
nentries = m - 1 - j;
memcpy (&nodeptr (0),&nodeptr (j),(nentries + 1)*sizeof (long));
memcpy (&recptr (0),&recptr (j),nentries*sizeof (long));
memcpy (entry (0),entry (j),nentries*length);
Write (indexfile,n2,node,NODESIZE);
nentries = 1;
nodeptr (0) = n1;
nodeptr (1) = n2;
recptr (0) = splitptr;
memcpy (entry (0),splitkey,length);
}
else
{
    memmove (&nodeptr (lo + 1),&nodeptr (lo),(nentries - lo + 1)*sizeof (long));
    memmove (&recptr (lo + 1),&recptr (lo),(nentries - lo)*sizeof (long));
    memmove (entry (lo + 1),entry (lo),(nentries - lo)*length);
    nodeptr (lo) = newnodeptr;
    recptr (lo) = newptr;
    memcpy (entry (lo),newkey,length);
    nentries++;
}
Write (indexfile,0,node,NODESIZE);
}
}
else /* Special case for no existing keys */
{
    memset (node,0xff,sizeof node);
    nentries = 1;
    recptr (0) = p;
    memcpy (entry (0),key,length);
}

```

```

    Write (indexfile,0,node,sizeof node);
}

if (listptr[2] < 0) /* If linked list is empty, initialize it */
{
    listptr[1] = p;
    listptr[2] = p;
}
else /* add record to linked list */
{
    ((long *)rec)[1] = listptr[2];
    Write (file,listptr[2],&p,sizeof (long));
    listptr[2] = p;
}

Write (file,p,rec,size);
Write (miscfile,misloff,listptr,sizeof listptr);
*ptr = p;
}

/* Find data record in file given key value */

void find (int file,int _indexfile,long *ptr,void *rec,int size,int _length,
char *key)
{
    int lo,hi,i,c;

    indexfile = _indexfile;
    length = _length;
    calcm ();

    *ptr = -1;

    if (filelength (indexfile) == 0) /* Special case for no keys in index */
        return;

    Read (indexfile,0,node,NODESIZE);

    for (;;)
    {
        /* Try to find key in current node */

        lo = 0;
        hi = nentries - 1;
        while (lo <= hi)
        {
            i = (lo + hi) / 2;

```

```

    c = memicmp (entry (i),key,length);
    if (c == 0) /* Found key, so read data record */
    {
        *ptr = recptr (i);
        Read (file,*ptr,rec,size);
        return;
    }
    if (c < 0)
        lo = i + 1;
    else
        hi = i - 1;
}

if (nodeptr (i) < 0) /* Bottom of tree, so give up */
    return;

    Read (indexfile,nodeptr (i),node,NODESIZE);
}
}

/* Get key value from user and try to find corresponding data record */

void select (int file,int _indexfile,long *ptr,void *rec,int size,char *key,
    int _length,int x,int y,int directions)
{
    indexfile = _indexfile;
    length = _length;

REDO:

    /* Ask user for key value */

    editstr (key,length,x,y,directions);

    if (dir == DIR_ESC) /* User hit ESC, so cancel */
        return;

    find (file,indexfile,ptr,rec,size,length,key);

    if (*ptr < 0) /* not found, so ask user for new key value */
    {
        beep ();
        goto REDO;
    }
}

/* Link a data record onto another one */

```

```

void relink (int fromfile,long fromptr,long *fromptrs,int fromoffset,
    int tofile,long toptr,long *toptrs,int tooffset)
{
    fromptrs[0] = toptr;
    fromptrs[2] = toptrs[1];
    if (toptrs[1] < 0)
        toptrs[0] = toptrs[1] = fromptr;
    else
    {
        Write (fromfile,toptrs[1] + fromoffset + sizeof (long),&fromptr,sizeof (long));
        toptrs[1] = fromptr;
    }
}

    /* Unlink a data record from another one */

void recunlink (int fromfile,long fromptr,long *fromptrs,int fromoffset,
    int tofile,long toptr,long *toptrs,int tooffset)
{
    if (toptrs[0] == fromptr)
        toptrs[0] = fromptrs[1];
    else
        Write (fromfile,fromptrs[2] + fromoffset + sizeof (long),&fromptrs[1],sizeof (long));
    if (toptrs[1] == fromptr)
        toptrs[1] = fromptrs[2];
    else
        Write (fromfile,fromptrs[1] + fromoffset + 2*sizeof (long),&fromptrs[2],sizeof (long));
    fromptrs[0] = fromptrs[1] = fromptrs[2] = -1;
}

    /* Open a file if it exists, otherwise create it, with error checking */

opencreate (char *filename)
{
    int f;

    f = open (filename,O_RDWR);
    if (f >= 0)
        return f;
    f = open (filename,O_CREAT | O_TRUNC | O_RDWR,0777);
    if (f < 0)
    {
        printf ("Can't create file %s.\n",filename);
        exit (1);
    }
    return f;
}

```

```

}

/* Functions to initialize data in miscfile */

void mwrite (void *buf,int bytes)
{
    if (write (miscfile,buf,bytes) != bytes)
    {
        printf ("Error writing to file misc.dat\n");
        exit (1);
    }
}

void mblank3 (void)
{
    static long data[] = { -1,-1,-1 };

    mwrite (data,sizeof data);
}

void mblank4 (void)
{
    static long data[] = { -1,-1,-1,-1 };

    mwrite (data,sizeof data);
}

```

## D.5 DPG.C

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include "video.h"
#include "files.h"

#define MAXNAME 32 /* Maximum length of symbols */
#define MAXMENU 20 /* Maximum depth of submenus */

#define myisalpha(c) (isalpha (c) || (c) == '_')
#define myisalnum(c) (isalpha (c) || (c) == '_' || isdigit (c))
#define addlist(b,p) (p)->next = (b); (b) = (p)
#define align(n) n = (n+1) & ~1

/* Types of variables */

```



```
enum
{
    V_INT,
    V_FLOAT,
    V_DATE,
    V_STR,
    V_PTR,
};

    /* Types of lexical tokens */
```

```
enum
{
    T_AND,
    T_ASSIGN,
    T_CLOSEBRACE,
    T_CLOSEBRACKET,
    T_COLON,
    T_DOT,
    T_EOF,
    T_EQ,
    T_FIELD,
    T_FILE,
    T_FLOAT,
    T_GE,
    T_GT,
    T_INT,
    T_KEYWORD,
    T_LE,
    T_LT,
    T_MINUS,
    T_NE,
    T_NOT,
    T_OPENBRACE,
    T_OPENBRACKET,
    T_OR,
    T_PERCENT,
    T_PLUS,
    T_PROC,
    T_SLASH,
    T_STAR,
    T_STR,
    T_UNDEF,
    T_VAR,
};
```

```
/* List of keywords */
```

```
enum
```

```
{  
    K_ATODATE,  
    K_ATOF,  
    K_ATOI,  
    K_BREAK,  
    K_CONTINUE,  
    K_CREATE,  
    K_CURSOR,  
    K_DATE,  
    K_DATEDAY,  
    K_DATEMONTH,  
    K_DATEYEAR,  
    K_DEL,  
    K_DELETE,  
    K_DO,  
    K_DOWN,  
    K_EDIT,  
    K_ELSE,  
    K_END,  
    K_ESC,  
    K_FILES,  
    K_FIND,  
    K_FLOAT,  
    K_GETKEY,  
    K_GOTO,  
    K_HOME,  
    K_IF,  
    K_INDEX,  
    K_INS,  
    K_INT,  
    K_LEFT,  
    K_LINK,  
    K_MAKEDATE,  
    K_ON,  
    K_PAUSE,  
    K_PGDN,  
    K_PGUP,  
    K_PRINTDATE,  
    K_PRINTFLOAT,  
    K_PRINTINT,  
    K_PRINTSTR,  
    K_PROCEDURES,  
    K_PTR,  
    K_READ,
```

```

K_RETURN,
K_RIGHT,
K_SCROLL,
K_SELECT,
K_SPRINTFDATE,
K_SPRINTFLOAT,
K_SPRINTINT,
K_STR,
K_STRCMP,
K_STRCPY,
K_SWITCH,
K_TABLE,
K_UNLINK,
K_UP,
K_VAR,
K_WHILE,
K_WRITE,
};

typedef char str[MAXNAME];

    /* Binary tree structure for symbol tables */

typedef struct symstruct
{
    struct symstruct *left,*right;
    str name;
    void *p; /* Pointer to actual object denoted by symbol */
} sym;

struct filestruct;
typedef struct filestruct file;

    /* Structure for file identifier ("name" field is required because
    pointer may not be available initially */

typedef struct fileidstruct
{
    struct fileidstruct *next;
    str name;
    file *p;
} fileid;

    /* Structure for variable declarations */

typedef struct varstruct
{

```

```

struct varstruct *next;
str name;
char type;
char l; /* Length in characters or digits before decimal point */
char dec; /* Digits after decimal point (V_FLOAT only) */
} var;

```

/\* Structure for data file \*/

```

struct filestruct
{
    file *next;
    str name;
    var *fields;
    sym *fieldnames;
    var index; /* Index field if any */
    fileid *on; /* Which files is this one linked on */
    long table; /* If nonzero, this file is a table and the value is the
                number of entries */
};

```

/\* Structure for procedure declaration \*/

```

typedef struct procstruct
{
    struct procstruct *next;
    str name;
    char *menuitem; /* Which menu item (if any) is used for access */
} proc;

```

/\* Structures used in constructing menu tree \*/

```

struct menutreestruct;
typedef struct menutreestruct menutree;

```

```

typedef struct submenustruct
{
    struct submenustruct *next;
    menutree *mt;
} submenu;

```

```

struct menutreestruct
{
    char *text;
    char *func;
    submenu *sm;
};

```

```
char *keywords[] =
{
    "atodate",
    "atof",
    "atoi",
    "break",
    "continue",
    "create",
    "cursor",
    "date",
    "dateday",
    "datemonth",
    "dateyear",
    "del",
    "delete",
    "do",
    "down",
    "edit",
    "else",
    "end",
    "esc",
    "files",
    "find",
    "float",
    "getkey",
    "goto",
    "home",
    "if",
    "index",
    "ins",
    "int",
    "left",
    "link",
    "makedate",
    "on",
    "pause",
    "pgdn",
    "pgup",
    "printdate",
    "printfloat",
    "printint",
    "printstr",
    "procedures",
    "ptr",
    "read",
    "return",
}
```

```

"right",
"scroll",
"select",
"sprintdate",
"sprintfloat",
"sprintint",
"str",
"strcmp",
"strcpy",
"switch",
"table",
"unlink",
"up",
"var",
"while",
"write",
};

/* Names of cursor directions used to move between fields
   (duplicated in keywords above) */

char *dirname[] =
{
    "RETURN",
    "ESC",
    "UP",
    "DOWN",
    "LEFT",
    "RIGHT",
    "PGUP",
    "PGDN",
    "HOME",
    "END",
    "INS",
    "DEL",
};

FILE *infile;      /* Source file */
FILE *outfile;     /* Output file */
char outfilename[256]; /* Name of output file */

long line = 1;     /* Current line number */
char buf[256];    /* Character buffer for lexical analyzer */

int nc;           /* Next character (used by lexical analyzer) */
int nt;           /* Next token type */
long ni;          /* Next integer value if integer token */

```

```

double nf;          /* Next floating point value if floating point token */
int nkeyword;      /* Next keyword if keyword token */
var *nvar;         /* Pointer to variable if variable name token */
file *nfile;       /* Pointer to file if file name token */
proc *nproc;       /* Pointer to procedure if procedure name token */
                  /* Labels for output switch statement depending on
                  cursor movement direction */
char dirlabel[MAXDIR][MAXNAME];
int indent;        /* Current output indentation level */
int lastlen,lastdec; /* Number of characters/digits of last token */

var *globals;     /* Global variables */
proc *procs;      /* Procedures */
file *files;      /* Files */
var *locals;      /* Local variables for procedure currently being parsed */

sym *globalnames;
sym *procnames;
sym *filenames;
sym *localnames;

long miscoff;     /* Offset for data in miscfile */

                  /* Text of menu items */
char *menutext[MAXMENU];

void parseexp (void);
void parsestatement (void);

                  /* Output error message and terminate */

void error (char *format,...)
{
    va_list argptr;

    va_start (argptr,format);
    printf ("Error %ld: ",line);
    vprintf (format,argptr);
    va_end (argptr);
    putchar ('\n');
    fclose (outfile);
    unlink (outfilename);
    exit (1);
}

                  /* Output error message but keep compiling */

```

```

void warning (char *format,...)
{
    va_list argptr;

    va_start (argptr,format);
    printf ("Warning %ld: ",line);
    vprintf (format,argptr);
    va_end (argptr);
    putchar ('\n');
}

    /* Allocate memory with error checking */

void *alloc (int n)
{
    void *p;

    p = malloc (n);
    if (p == 0)
    {
        error ("Out of memory");
        exit (1);
    }
    return p;
}

    /* Add a symbol to a symbol table */

void addsym (char *name,sym **base,void *p)
{
    sym *s;
    sym *q;
    int c;

    s = alloc (sizeof (sym));
    s->left = s->right = 0;
    strcpy (s->name,name);
    s->p = p;
    if (*base == 0)
    {
        *base = s;
        return;
    }
    q = *base;
    for (;;)
    {
        c = strcmp (q->name,name);

```



```

if (c == 0)
{
    warning ("Symbol '%s' redefined",name);
    return;
}
if (c < 0)
{
    if (q->right == 0)
    {
        q->right = s;
        return;
    }
    q = q->right;
}
else
{
    if (q->left == 0)
    {
        q->left = s;
        return;
    }
    q = q->left;
}
}
}

/* Free a symbol table */

void freesym (sym *s)
{
    if (s)
    {
        freesym (s->right);
        freesym (s->left);
        free (s);
    }
}

/* Compare pointers to pointers to strings (used in calling bsearch ()) */

strcmp (char **s1,char **s2)
{
    return strcmp (*s1,*s2);
}

/* Locate the string in buf in a symbol table, if present */

```

```

void *findsym (sym *s)
{
    int i;

    if (s == 0)
        return 0;
    i = strcmp (buf,s->name);
    if (i == 0)
        return s->p;
    if (i < 0)
        return findsym (s->left);
    else
        return findsym (s->right);
}

    /* Read next character into nc from input file */

```

```

void readc (void)
{
    nc = fgetc (infile);
    if (nc == '\n')
        line++;
}

```

/\* Read next lexical token \*/

```

void lex (void)
{
    int i;
    void *p;

```

LOOP:

/\* Ignore leading white space \*/

```

while (isspace (nc))
    readc ();

```

/\* Next token is a symbol of some sort \*/

```

if (myisalpha (nc))
{
    /* Read symbol into buf */

    i = 0;
    do
    {

```

```

    if (i != MAXNAME)
        buf[i++] = tolower (nc);
    readc ();
}
while (myisalnum (nc));
buf[i] = 0;

    /* Is it a keyword? */

p = buf;
p = bsearch (&p,keywords,sizeof keywords / sizeof (char *),
    sizeof (char *),strcmp);
if (p)
{
    nt = T_KEYWORD;
    nkeyword = (char **)p - keywords;
    return;
}

    /* Is it the name of a local variable? */

p = findsym (localnames);
if (p)
{
    nt = T_VAR;
    nvar = p;
    lastlen = nvar->l;
    lastdec = nvar->dec;
    return;
}

    /* Is it the name of a global variable? */

p = findsym (globalnames);
if (p)
{
    nt = T_VAR;
    nvar = p;
    lastlen = nvar->l;
    lastdec = nvar->dec;
    return;
}

    /* Is it the name of a file? */

p = findsym (filenames);
if (p)

```

```

{
    nt = T_FILE;
    nfile = p;
    return;
}

    /* Is it the name of a procedure? */

p = findsym (procnames);
if (p)
{
    nt = T_PROC;
    nproc = p;
    return;
}

    /* Failing all this, if the last two tokens were the name of a file
       followed by a dot, this must be the name of a field */

if (nt == T_DOT && nfile)
{
    /* Is it a regular field? */

    nt = T_FIELD;
    p = findsym (nfile->fieldnames);
    if (p)
    {
        nvar = p;
        lastlen = nvar->l;
        lastdec = nvar->dec;
    }
    else /* Is it the index variable? */
        if (!strcmp (buf,nfile->index.name))
        {
            nvar = &nfile->index;
            lastlen = nvar->l;
        }
    return;
}

    /* Not identified, so return next token as unidentified symbol */

nt = T_UNDEF;
return;
}

    /* Next token is a number */

```

```

if (isdigit (nc))
{
    /* Read number into buf */

    i = 0;
    do
    {
        buf[i] = nc;
        i++;
        if (i == sizeof buf)
            error ("Number too long");
        readc ();
    }
    while (isdigit (nc));
    lastlen = i;

    /* Floating point number */

    if (nc == '.')
    {
        buf[i] = '.';
        i++;
        if (i == sizeof buf)
            error ("Number too long");
        readc ();
        while (isdigit (nc))
        {
            buf[i] = nc;
            i++;
            if (i == sizeof buf)
                error ("Number too long");
            readc ();
        }
        buf[i] = 0;
        nt = T_FLOAT;
        nf = atof (buf);
        lastdec = i - lastlen - 1;
    }
    else /* Integer */
    {
        buf[i] = 0;
        nt = T_INT;
        ni = atol (buf);
    }
    return;
}

```

```

switch (nc)
{
case '!': /* Either a floating-point number or the dot operator */
    readc ();
    if (isdigit (nc))
    {
        buf[0] = '!';
        i = 1;
        do
        {
            buf[i] = nc;
            i++;
            if (i == sizeof buf)
                error ("Number too long");
        }
        while (isdigit (nc));
        buf[i] = 0;
        nt = T_FLOAT;
        nf = atof (buf);
        lastlen = 0;
        lastdec = i - 1;
    }
    else
        nt = T_DOT;
    break;
case '"': /* A string */
    i = 0;
    readc ();
    while (nc != '"')
    {
        if (nc == EOF)
            error ("Unterminated string");
        buf[i] = nc;
        i++;
        if (i == sizeof buf)
            error ("String too long");
        readc ();
    }
    buf[i] = 0;
    lastlen = i;
    readc ();
    nt = T_STR;
    break;
case '&':
    readc ();
    if (nc != '&')
        error ("'"&' is not a valid token");
}

```

```

    readc ();
    nt = T_AND;
    break;
case '|':
    readc ();
    if (nc != '|')
        error ("'| is not a valid token");
    readc ();
    nt = T_OR;
    break;
case '=':
    readc ();
    nt = T_ASSIGN;
    if (nc == '=')
    {
        readc ();
        nt = T_EQ;
    }
    break;
case '>':
    readc ();
    nt = T_GT;
    if (nc == '>')
    {
        readc ();
        nt = T_GE;
    }
    break;
case '<':
    readc ();
    nt = T_LT;
    if (nc == '<')
    {
        readc ();
        nt = T_LE;
    }
    break;
case '!':
    readc ();
    nt = T_NOT;
    if (nc == '!')
    {
        readc ();
        nt = T_NE;
    }
    break;
case '{':

```

```

    readc ();
    nt = T_OPENBRACE;
    break;
case '}':
    readc ();
    nt = T_CLOSEBRACE;
    break;
case '(':
    readc ();
    nt = T_OPENBRACKET;
    break;
case ')':
    readc ();
    nt = T_CLOSEBRACKET;
    break;
case ':':
    readc ();
    nt = T_COLON;
    break;
case '-':
    readc ();
    nt = T_MINUS;
    break;
case '%':
    readc ();
    nt = T_PERCENT;
    break;
case '+':
    readc ();
    nt = T_PLUS;
    break;
case '/': /* Either a slash or a comment */
    readc ();
    if (nc == '*')
    {
        do
        {
            do
                readc ();
            while (nc != '*' && nc != EOF);
            if (nc == EOF)
            {
                nt = T_EOF;
                return;
            }
        }
        readc ();
    }
}

```



```

        while (nc != '/');
        readc ();
        goto LOOP; /* Comment, so ignore it and get new token */
    }
    nt = T_SLASH;
    break;
case '*':
    readc ();
    nt = T_STAR;
    break;
case EOF:
    nt = T_EOF;
    break;
default:
    error ("Unrecognized character %c",nc);
}
}

/* Check whether the next token is a given keyword, if so get new token */

iskeyword (int k)
{
    if (nt == T_KEYWORD && nkeyword == k)
    {
        lex ();
        return 1;
    }
    return 0;
}

/* Output a string */

void os (char *s)
{
    fprintf (outfile,"%s",s);
}

/* Output an integer */

void oi (long n)
{
    fprintf (outfile,"%ld",n);
}

/* Output a floating point number */

void of (double n)

```

```

{
    fprintf (outfile,"%lf",n);
}

    /* Go on to new line, and indent as necessary */

void nl (void)
{
    int i;

    fputc ('\n',outfile);
    for (i=indent; i--;)
        fputc ('\t',outfile);
}

    /* Check that the next token is a label reference (labels are not
       currently validated, so pass any sort of symbol */

void cklabel (void)
{
    if ( nt != T_UNDEF &&
        nt != T_KEYWORD &&
        nt != T_VAR &&
        nt != T_FILE &&
        nt != T_PROC &&
        nt != T_FIELD)
        error ("Label expected");
}

    /* Check that the next token is a file name */

void ckfile (void)
{
    if (nt != T_FILE)
        error ("File name expected");
}

    /* Check that the next token is the appropriate keyword, and if so, get
       new token */

void parsekeyword (int k)
{
    if (!iskeyword (k))
    {
        strcpy (buf,keywords[k]);
        strupr (buf);
        error ("%s expected",buf);
    }
}

```

```
}  
}
```

```
/* Check that the next token is an integer constant, and if so, return  
its value and get new token */
```

```
long parseconst (void)
```

```
{  
    long n;  
  
    if (nt != T_INT)  
        error ("Integer expected");  
    n = ni;  
    lex ();  
    return n;  
}
```

```
/* Parse a list of variable declarations */
```

```
void parsevarlist (var **vl,sym **s)
```

```
{  
    var *v;  
  
    while (nt == T_KEYWORD &&  
           ( nkeyword == K_INT ||  
             nkeyword == K_FLOAT ||  
             nkeyword == K_DATE ||  
             nkeyword == K_STR ||  
             nkeyword == K_PTR))  
    {  
        v = alloc (sizeof (var));  
        switch (nkeyword)  
        {  
            case K_INT:  
                lex ();  
                v->type = V_INT;  
                v->l = parseconst ();  
                break;  
            case K_FLOAT:  
                lex ();  
                v->type = V_FLOAT;  
                v->l = parseconst ();  
                v->dec = parseconst ();  
                break;  
            case K_DATE:  
                lex ();  
                v->type = V_DATE;
```

```

        break;
    case K_STR:
        lex ();
        v->type = V_STR;
        v->l = parseconst ();
        break;
    case K_PTR:
        lex ();
        v->type = V_PTR;
        break;
    }
    strcpy (v->name,buf);
    lex ();
    addsym (v->name,s,v);
    addlist (*vl,v);
}
}

/* Routines for top-down expression parse */

```

```

void parsefactor (void)
{
    switch (nt)
    {
        case T_MINUS:
            os ("-");
            lex ();
            parsefactor ();
            break;
        case T_NOT:
            os ("!");
            lex ();
            parsefactor ();
            break;
        case T_INT:
            oi (ni);
            lex ();
            break;
        case T_FLOAT:
            of (nf);
            lex ();
            break;
        case T_STR:
            os ("\"");
            os (buf);
            os ("\"");
            lex ();
    }
}

```

```

    break;
case T_OPENBRACKET:
    os ("(");
    lex ();
    parseexp ();
    if (nt != T_CLOSEBRACKET)
        error ("')' expected");
    lex ();
    os (")");
    break;
case T_VAR:
    os (buf);
    lex ();
    break;
case T_FILE:
    os (buf);
    lex ();
    if (nt != T_DOT)
        error (".' expected");
    lex ();
    os (".");
    if (nt != T_FIELD)
        error ("Field name expected");
    os (buf);
    lex ();
    break;
case T_KEYWORD:
    switch (nkeyword)
    {
        case K_ATODATE:
        case K_ATOF:
        case K_ATOI:
        case K_DATEDAY:
        case K_DATEMONTH:
        case K_DATEYEAR:
            os (buf);
            os (" (");
            lex ();
            parseexp ();
            os (")");
            break;
        case K_MAKEDATE:
            lex ();
            os ("makedate (");
            parseexp ();
            os (",");
            parseexp ();

```

```

        os (",");
        parseexp ();
        os ("");
        break;
    case K_STRCMP:
        lex ();
        os ("memicmp (");
        parseexp ();
        os (",");
        parseexp ();
        os (",");
        if (nt == T_INT)
            parseexpr ();
        else
            oi (lastlen);
        os ("");
        break;
    default:
        error ("Expression expected");
}
break;
default:
    error ("Expression expected");
}
}

void parseterm (void)
{
    parsefactor ();
    while (nt == T_STAR || nt == T_SLASH || nt == T_PERCENT)
    {
        switch (nt)
        {
            case T_STAR:
                os ("*");
                break;
            case T_SLASH:
                os ("/");
                break;
            case T_PERCENT:
                os ("%");
                break;
        }
        lex ();
        parsefactor ();
    }
}

```

```

void parsemathexp (void)
{
    parseterm ();
    while (nt == T_PLUS || nt == T_MINUS)
    {
        if (nt == T_PLUS)
            os (" + ");
        else
            os (" - ");
        lex ();
        parseterm ();
    }
}

```

```

void parserelexp (void)
{
    parsemathexp ();
    while (nt == T_GT || nt == T_LT || nt == T_GE || nt == T_LE)
    {
        switch (nt)
        {
            case T_GT:
                os (" > ");
                break;
            case T_LT:
                os (" < ");
                break;
            case T_GE:
                os (" >= ");
                break;
            case T_LE:
                os (" <= ");
                break;
        }
        lex ();
        parsemathexp ();
    }
}

```

```

void parseeqexp (void)
{
    parserelexp ();
    while (nt == T_EQ || nt == T_NE)
    {
        if (nt == T_EQ)
            os (" == ");
    }
}

```

```

    else
        os (" != ");
    lex ();
    parserelexp ();
}
}

```

```

void parseandexp (void)
{
    parseeqexp ();
    while (nt == T_AND)
    {
        os (" && ");
        lex ();
        parseeqexp ();
    }
}

```

```

void parseexp (void)
{
    parseandexp ();
    while (nt == T_OR)
    {
        os (" || ");
        lex ();
        parseandexp ();
    }
}

```

/\* Parse a statement, or zero or more statements surrounded by braces,  
with appropriate output indentation \*/

```

void parseindentstatement (void)
{
    if (nt == T_OPENBRACE)
        parsestatement ();
    else
    {
        indent++;
        parsestatement ();
        indent--;
    }
}

```

/\* Parse the section of a PRINT statement common to all data types \*/

```

void parseprint (void)

```



```

{
lex ();
parseexp (); /* X */
os (",");
parseexp (); /* Y */
os (",");
parseexp (); /* Expression to be displayed */
}

```

/\* Parse the section of an SPRINT statement common to all data types \*/

```

void parsesprint (void)
{
lex ();
parseexp (); /* String variable */
os (",");
parseexp (); /* Expression to be printed */
}

```

/\* Parse a direction keyword for an EDIT statement, optionally followed by a label reference \*/

```

void parsedirection (void)
{
int i;

if (nt != T_KEYWORD)
error ("Direction expected");
for (i=0; i!=MAXDIR; i++)
if (!strcmp (dirname[i],buf))
{
lex ();
dirlabel[i][0] = '*';
if (nt == T_ASSIGN) /* If label reference present, record it */
{
lex ();
cklabel ();
strupr (buf);
strcpy (dirlabel[i],buf);
lex ();
}
return;
}
error ("Direction expected");
}

```

/\* Parse a direction for an EDIT statement, or zero or more directions

```

        surrounded by braces */

void parsedirections (void)
{
    int i;
    int ndir;

    os (",");

    /* Read direction data */

    memset (dirlabel,0,sizeof dirlabel);
    if (nt == T_OPENBRACE)
    {
        lex ();
        while (nt != T_CLOSEBRACE)
            parsedirection ();
        lex ();
    }
    else
        parsedirection ();

    /* Output flags indicating allowed directions */

    ndir = 0;
    for (i=0; i!=MAXDIR; i++)
        if (dirlabel[i][0])
            ndir++;
    for (i=0; i!=MAXDIR; i++)
        if (dirlabel[i][0])
        {
            os ("DF_");
            os (dirname[i]);
            ndir--;
            if (ndir)
                os (" | ");
        }
    os ("");

    /* Output switch statement on direction */

    nl ();
    os ("switch (dir)");
    nl ();
    os ("{");
    indent++;
    for (i=0; i!=MAXDIR; i++)

```

```

if (dirlabel[i][0] != 0 && dirlabel[i][0] != '*')
{
    nl ();
    os ("case DIR_");
    os (dirname[i]);
    os (":");
    indent++;
    nl ();
    os ("goto ");
    os (dirlabel[i]);
    os (";");
    indent--;
}
indent--;
nl ();
os ("}");
}

```

/\* Parse an EDIT statement \*/

```

void parseedit (void)
{
    switch (nvar->type)
    {
        case V_INT:
            switch (nvar->l)
            {
                case 1:
                case 2:
                    os ("editbyte (&");
                    break;
                case 3:
                case 4:
                    os ("editshort (&");
                    break;
                default:
                    os ("editlong (&");
            }
            break;
        case V_FLOAT:
            os ("editfloat (&");
            break;
        case V_DATE:
            os ("editdate (&");
            break;
        case V_STR:
            os ("editstr (");

```

```

        break;
    case V_PTR:
        warning ("Can't edit a pointer variable");
        break;
    }
    if (nt == T_FIELD)
    {
        if (nvar == &nfile->index)
            warning ("Can't edit index field");
        os (nfile->name);
        os (".");
    }
    os (buf);
    if (nvar->type != V_DATE)
    {
        os ("," );
        oi (nvar->l);
        if (nvar->type == V_FLOAT)
        {
            os ("," );
            oi (nvar->dec);
        }
    }
    lex ();
    os ("," );
    parseexp ();
    os ("," );
    parseexp ();
    parsedirections ();
}

/* Parse a statement */

void parsestatement (void)
{
    file *f,*f2;
    int x,y,i;
    str vname;

    switch (nt)
    {
        case T_OPENBRACE: /* Zero or more statements surrounded by braces */
            nl ();
            os ("{" );
            lex ();
            indent++;
            while (nt != T_CLOSEBRACE)

```

```

    parsestatement ();
    indent--;
    nl ();
    os ("}");
    lex ();
    break;
case T_COLON:    /* A label */
    lex ();
    cklabel ();
    strupr (buf);
    fprintf (outfile, "\n\n%s:", buf);
    lex ();
    if (nt == T_CLOSEBRACE)
        os (";");
    break;
case T_VAR:     /* Assignment to a variable */
    nl ();
    os (buf);
    lex ();
    if (nt != T_ASSIGN)
        error ("=' expected");
    lex ();
    os (" = ");
    parseexp ();
    os (";");
    break;
case T_FILE:   /* Assignment to a field */
    nl ();
    os (buf);
    lex ();
    if (nt != T_DOT)
        error (".' expected");
    lex ();
    os (".");
    if (nt != T_FIELD)
        error ("Field name expected");
    os (buf);
    if (nt != T_ASSIGN)
        error ("=' expected");
    lex ();
    os (" = ");
    parseexp ();
    os (";");
    break;
case T_PROC:   /* A procedure call */
    nl ();
    os (buf);

```

```

os (" (");
lex ();
break;
case T_KEYWORD:
switch (nkeyword)
{
case K_EDIT: /* Edit a variable or field */
lex ();
nl ();
switch (nt)
{
case T_VAR:
parseedit ();
break;
case T_FILE:
lex ();
if (nt != T_DOT)
error (". expected");
lex ();
if (nt != T_FIELD)
error ("Field name expected");
parseedit ();
break;
default:
error ("Variable name expected");
}
break;
case K_SELECT: /* Allow user to select a data record */
lex ();
ckfile ();
nl ();
os ("select (");
os (buf);
os ("file,");
os (buf);
os ("indexfile,&");
os (buf);
os ("ptr,&");
os (buf);
os (",sizeof ");
os (buf);
os (",");
os (buf);
os (".");
os (nfile->index.name);
os (",");
oi (nfile->index.l);

```

```

    os (",");
    lex ();
    parseexp ();
    os (",");
    parseexp ();
    parsedirections ();
    break;
case K_FIND:    /* Find a data record given the index value */
    lex ();
    ckfile ();
    nl ();
    os ("find (");
    os (buf);
    os ("file,");
    os (buf);
    os ("indexfile,&");
    os (buf);
    os (",sizeof ");
    os (buf);
    os (",");
    oi (nfile->index.l);
    os (",");
    lex ();
    parseexp ();
    os (");");
    break;
case K_READ:   /* Read a data record */
    lex ();
    ckfile ();
    nl ();
    os ("Read (");
    f = nfile;
    if (f->table)
    {
        os ("miscfile,");
        os (buf);
        os ("miscoff + (");
        lex ();
        parseexp ();
        os (")*sizeof ");
        os (f->name);
        os (",&");
        os (f->name);
        os (",sizeof ");
        os (f->name);
        os (");");
    }
}

```

```

else
{
  os (buf);
  os ("file,");
  if (nt == T_VAR)
  {
    if (nvar->type != V_PTR)
      warning ("Variable '%s' is not of type PTR",buf);
    strcpy (vname,buf);
    lex ();
    os (vname);
    os (",&");
    os (f->name);
    os (",sizeof ");
    os (f->name);
    os ("");
    nl ();
    os (f->name);
    os ("ptr = ");
  }
  else
  {
    if (nt != T_FILE)
      error ("File read location must be a variable of type PTR");
    f2 = nfile;
    lex ();
    if (nt != T_DOT)
      error (". expected");
    lex ();
    if (nt != T_FIELD)
      error ("Field name expected");
    strcpy (vname,buf);
    lex ();
    os (f2->name);
    os (".");
    os (vname);
    os (",&");
    os (f->name);
    os (",sizeof ");
    os (f->name);
    os ("");
    nl ();
    os (f->name);
    os ("ptr = ");
    os (f2->name);
    os (".");
  }
}

```



```

    os (vname);
    os (";");
}
break;
case K_WRITE: /* Write back currently selected record to file */
lex ();
ckfile ();
nl ();
os ("Write (");
f = nfile;
if (f->table)
{
    os ("miscfile,");
    os (buf);
    os ("miscoff + (");
    lex ();
    parseexp ();
    os (")*sizeof ");
    os (f->name);
    os (",&");
    os (f->name);
    os ("sizeof ");
    os (f->name);
    os (");");
}
else
{
    os (buf);
    os ("file,");
    os (buf);
    os ("ptr,&");
    os (buf);
    os ("sizeof ");
    os (buf);
    os (");");
    lex ();
}
break;
case K_CREATE: /* Create a new record */
lex ();
x = -1;
if (nt == T_INT)
{
    x = parseconst ();
    y = parseconst ();
}
ckfile ();

```

```

if (nfile->table)
    warning ("Can't create record on table");
if (x >= 0 && !nfile->index.l)
    warning ("X, Y parameters should only be provided for creation on indexed files");
if (x < 0 && nfile->index.l)
    warning ("X, Y parameters should be provided for creation on indexed files");
nl ();
os (buf);
os (" = blank");
os (buf);
os (";");
nl ();
os ("create");
if (x >= 0)
    os ("index");
os (" (");
os (buf);
os ("file,&");
os (buf);
os ("ptr,&");
os (buf);
os (" ,sizeof ");
os (buf);
os (" ,");
os (buf);
os ("miscoff");
if (x >= 0) /* File is indexed, so index value must be input */
{
    os (" ,");
    os (buf);
    os (" .");
    os (nfile->index.name);
    os (" ,");
    oi (nfile->index.l);
    os (" ,");
    os (buf);
    os ("indexfile,");
    oi (x);
    os (" ,");
    oi (y);
    lex ();
    parsedirections ();
}
else
{
    os (";");
    lex ();
}

```

```

}
break;
case K_DELETE: /* Delete currently selected record */
lex ();
ckfile ();
if (nfile->table)
warning ("Can't delete record from table");
nl ();
if (nfile->index.l)
os ("deleteindex (");
else
os ("delete (");
os (buf);
os ("file,");
os (buf);
os ("ptr,&");
os (buf);
os (",");
os (buf);
os ("miscoff");
if (nfile->index.l)
{
os (",");
os (buf);
os (".");
os (nfile->index.name);
os (",");
oi (nfile->index.l);
os (",");
os (buf);
os ("indexfile");
}
os ("");
lex ();
break;
case K_LINK:
case K_UNLINK:
nl ();
os ("rec");
os (buf);
os (" (");
lex ();
ckfile ();
os (buf);
os ("file,");
os (buf);
os ("ptr,&");

```

```

os (buf);
os (".");
f = nfile;
lex ();
ckfile ();
os (buf);
os ("ptr,foffset (");
os (f->name);
os (",");
os (buf);
os ("ptr,");
os (buf);
os ("file,");
os (buf);
os ("ptr,&");
os (buf);
os (".");
os (f->name);
os ("first,foffset (");
os (buf);
os (",");
os (f->name);
os ("first));");
lex ();
break;
case K_SCROLL: /* Scrolling edit */
lex ();
nl ();
os ("scrolledit (");
ckfile ();
os (buf);
os ("file,&");
os (buf);
os ("ptr,&");
os (buf);
os (",foffset (");
os (buf);
os (",");
f = nfile;
lex ();
ckfile ();
os (buf);
os ("ptr,sizeof ");
os (f->name);
os (",");
os (buf);
os ("file,");

```

```

os (buf);
os ("ptr,&");
os (buf);
os (" ,foffset (");
os (buf);
os (" ,");
os (f->name);
os ("first),sizeof ");
os (buf);
os (" ,&blank");
os (f->name);
os (" ,");
os (f->name);
os ("miscoff,");
lex ();
parseexp ();
os (" ,");
if (nt != T_PROC)
    error ("Procedure name expected");
os (buf);
os (" ,");
lex ();
switch (nt)
{
    case T_PROC:
        os (buf);
        lex ();
        break;
    case T_INT:
        if (parseconst () == 0)
        {
            os ("0");
            break;
        }
    default:
        error ("Procedure name or zero expected");
}
os (" ,");
parseexp ();
os (" ,");
parseexp ();
os (" ,");
parseexp ();
os (" ,");
parseexp ();
os (" ,");
parseexp ();
os (" ,");
break;

```

```

case K_GOTO:    /* Jump to a label */
    lex ();
    cklabel ();
    nl ();
    os ("goto ");
    strupr (buf);
    os (buf);
    os ("");
    break;
case K_BREAK:
case K_CONTINUE:
case K_RETURN:
    nl ();
    os (buf);
    os ("");
    lex ();
    break;
case K_IF:     /* C-style IF */
    nl ();
    os ("if (");
    lex ();
    parseexp ();
    os ("");
    parseindentstatement ();
    if (iskeyword (K_ELSE))
    {
        nl ();
        os ("else");
        parseindentstatement ();
    }
    break;
case K_WHILE:  /* C-style WHILE */
    nl ();
    os ("while (");
    lex ();
    parseexp ();
    os ("");
    parseindentstatement ();
    break;
case K_DO:     /* C-style DO...WHILE */
    nl ();
    os ("do");
    lex ();
    parseindentstatement ();
    parsekeyword (K_WHILE);
    nl ();
    os ("while (");

```

```

    parseexp ();
    os ("");
    break;
case K_SWITCH: /* Limited CASE statement - list of labels to
                jump to on values of an integer */
    nl ();
    os ("switch (");
    lex ();
    parseexp ();
    os ("");
    nl ();
    os ("{");
    indent++;
    if (nt != T_OPENBRACE)
        error ("'{ expected");
    lex ();
    i = 0;
    while (nt != T_CLOSEBRACE)
    {
        cklable ();
        nl ();
        os ("case ");
        oi (i);
        i++;
        os (":");
        indent++;
        nl ();
        os ("goto ");
        strupr (buf);
        os (buf);
        os (");");
        lex ();
        indent--;
    }
    lex ();
    indent--;
    nl ();
    os ("}");
    break;
case K_PRINTINT: /* Display an integer expression */
    nl ();
    os ("printint (");
    parseprint ();
    os (",");
    if (nt == T_INT)
        parseexp ();
    else

```

```

    oi (lastlen);
os ("");
break;
    /* Display a floating-point expression */
case K_PRINTFLOAT:
    nl ();
    os ("printfloat (");
    parseprint ();
    os (",");
    if (nt == T_INT)
    {
        parseexp ();
        os (",");
        parseexp ();
    }
    else
    {
        oi (lastlen);
        os (",");
        oi (lastdec);
    }
    os ("");
    break;
case K_PRINTDATE: /* Display a date expression */
    nl ();
    os ("printdate (");
    parseprint ();
    os ("");
    break;
case K_PRINTSTR: /* Display a string expression */
    nl ();
    os ("printstr (");
    parseprint ();
    os (",");
    if (nt == T_INT)
        parseexp ();
    else
        oi (lastlen);
    os ("");
    break;
case K_SPRINTINT: /* Convert an integer expression to ASCII */
    nl ();
    os ("sprintint (");
    parsesprint ();
    os (",");
    if (nt == T_INT)
        parseexp ();

```



```

else
    oi (lastlen);
os ("");
break;
        /* Convert a floating point expression to ASCII */
case K_SPRINTFLOAT:
    nl ();
    os ("sprintfloat (");
    parsesprint ();
    os (",");
    if (nt == T_INT)
    {
        parseexp ();
        os (",");
        parseexp ();
    }
    else
    {
        oi (lastlen);
        os (",");
        oi (lastdec);
    }
    os ("");
    break;
        /* Convert a date expression to ASCII */
case K_SPRINTDATE:
    nl ();
    os ("sprintdate (");
    parsesprint ();
    os ("");
    break;
case K_CURSOR: /* Move the hardware cursor to a point on screen */
    lex ();
    nl ();
    os ("cursor (");
    parseexp ();
    os (",");
    parseexp ();
    os ("");
    break;
case K_PAUSE: /* Wait for a keypress */
    lex ();
    nl ();
    os ("getkey ()");
    break;
case K_GETKEY: /* Wait for a keypress, and put it into a string */
    lex ();

```

```

    nl ();
    parseexpr ();
    os ("[0] = getkey (");
    break;
case K_STRCPY: /* Copy one string to another */
    lex ();
    nl ();
    os ("memcpy (");
    parseexpr ();
    os (",");
    parseexpr ();
    os (",");
    if (nt == T_INT)
        parseexpr ();
    else
        oi (lastlen);
    os (");");
    break;
default:
    error ("Statement expected");
}
break;
default:
    error ("Statement expected");
}
}

```

/\* Output C code variable declarations \*/

```

void outvarlist (var *v)
{
    while (v)
    {
        nl ();
        switch (v->type)
        {
            case V_INT:
                switch (v->l)
                {
                    case 1:
                    case 2:
                        os ("signed char ");
                        break;
                    case 3:
                    case 4:
                        os ("short ");
                        break;
                }
            }
        }
    }
}

```

```

        default:
            os ("long ");
        }
        os (v->name);
        os ("");
        break;
    case V_FLOAT:
        os ("double ");
        os (v->name);
        os ("");
        break;
    case V_DATE:
        os ("unsigned short ");
        os (v->name);
        os ("");
        break;
    case V_STR:
        os ("char ");
        os (v->name);
        os ("[");
        oi (v->l);
        os ("]");
        break;
    case V_PTR:
        os ("long ");
        os (v->name);
        os ("");
        break;
    }
    v = v->next;
}
}

```

/\* Output null pointer value \*/

```

void blankptr (void)
{
    nl ();
    os ("-1,");
}

```

/\* Output null value for a variable \*/

```

void outblank (var *v)
{
    int i;

```

```

nl ();
switch (v->type)
{
  case V_DATE:
    os ("BLANKDATE,");
    break;
  case V_STR:
    for (i=v->l; i--;)
      os (" ",");
    break;
  default:
    os ("0,");
}
}

```

/\* Determine physical size of a data record \*/

```

reclsize (file *f)
{
  int n;
  file *f2;
  var *v;
  fileid *fi;

  /* Index field */

  n = f->index.l;

  /* Normal fields */

  for (v=f->fields; v; v=v->next)
    switch (v->type)
    {
      case V_INT:
        switch (v->l)
        {
          case 1:
          case 2:
            n++;
            break;
          case 3:
          case 4:
            align (n);
            n += sizeof (short);
            break;
          default:
            align (n);

```

```

        n += sizeof (long);
    }
    break;
case V_FLOAT:
    align (n);
    n += sizeof (double);
    break;
case V_DATE:
    align (n);
    n += sizeof (short);
    break;
case V_STR:
    n += v->l;
    break;
}

align (n);

/* Files this one is linked on */

for (fi=f->on; fi; fi=fi->next)
    n += 3*sizeof (long);

/* Files that are linked on this one */

for (f2=files; f2; f2=f2->next)
    for (fi=f2->on; fi; fi=fi->next)
        if (fi->p == f)
        {
            n += 2*sizeof (long);
            break;
        }

return n;
}

/* Functions to output menu structure */

void addmenu (menutree *p,int i,char *func)
{
    menutree *mt;
    submenu *sm;

    for (sm=p->sm; sm; sm=sm->next)
        if (!strcmp (sm->mt->text,menutext[i]))
        {
            mt = sm->mt;

```

```

        break;
    }
    if (sm)
    {
        i++;
        if (menutext[i] == 0)
            error ("Duplicate menu option '%s'",func);
        addmenu (mt,i,func);
    }
    else
    do
    {
        mt = alloc (sizeof (menutree));
        mt->text = menutext[i];
        mt->func = func;
        mt->sm = 0;
        sm = alloc (sizeof (submenu));
        sm->mt = mt;
        addlist (p->sm,sm);
        p = mt;
        i++;
    }
    while (menutext[i]);
}

void addoption (menutree *p,char *text,char *func)
{
    int i;
    char *s;

    i = 0;
    s = strtok (text,"!");
    do
    {
        menutext[i++] = s;
        if (i == MAXMENU)
            error ("Menu tree too deep: '%s'",text);
        s = strtok (0,"!");
    }
    while (s);
    menutext[i] = 0;
    addmenu (p,0,func);
}

void menuoutput (menutree *p,char *text)
{
    char nexttext[4 + 1 + MAXMENU*2];

```

```

int i;
submenu *sm;

for (sm=p->sm,i=0; sm; sm=sm->next,i++)
    if (sm->mt->sm)
        {
            sprintf (nexttext,"%s%02d",text,i);
            menuoutput (sm->mt,nexttext);
        }
nl ();
os ("execmenuitem ");
os (text);
os ("[] =");
nl ();
os ("{}");
indent = 1;
for (sm=p->sm,i=0; sm; sm=sm->next,i++)
    {
        nl ();
        os ("\\");
        os (sm->mt->text);
        os ("\\,");
        if (sm->mt->sm)
            {
                sprintf (nexttext,"%s%02d",text,i);
                os (nexttext);
                os ("0,");
            }
        else
            {
                os ("0,");
                os (sm->mt->func);
                os (",");
            }
    }
nl ();
os ("0");
indent = 0;
nl ();
os ("};");
nl ();
}

main (int argc,char **argv)
{
    var *v,*v2;
    file *f,*f2;

```

```

proc *p;
fileid *fi;
static menutree mt;

printf ("Database Program Generator v0.0 by Russell Wallace " __DATE__ "\n");

if (argc != 2)
{
    printf ("Usage: dpg filename\n");
    return 1;
}

    /* Get output file name */

strcpy (outfilename,argv[1]);
if (!strchr (outfilename,'.'))
    strcat (outfilename, ".dpg");

    /* Open input file */

infile = fopen (outfilename,"r");
if (infile == 0)
{
    printf ("Can't open '%s'\n",outfilename);
    return 1;
}

    /* Create output file */

strcpy (strchr (outfilename,'.'),".c");
outfile = fopen (outfilename,"w");
if (outfile == 0)
{
    printf ("Can't create '%s'\n",outfilename);
    return 1;
}

    /* Initialize lexical analyzer */

readc ();
lex ();

    /* Standard stuff for output file */

os ("#include<t<stdio.h>");
nl ();
os ("#include<t<stdlib.h>");

```



```

nl ();
os ("#include<string.h>");
nl ();
os ("#include<fcntl.h>");
nl ();
os ("#include<io.h>");
nl ();
os ("#include\t\"video.h\"");
nl ();
os ("#include\t\"files.h\"");
nl ();

    /* Global variables */

v = alloc (sizeof (var));
strcpy (v->name,"fieldno");
v->type = V_INT;
v->l = 4;
addsym ("fieldno",&globalnames,v);
addlist (globals,v);

parsevarlist (&globals,&globalnames);

outvarlist (globals);
if (globals)
    nl ();

    /* File definitions */

parsekeyword (K_FILES);

while (nt == T_UNDEF)
{
    /* Create file structure */

    f = alloc (sizeof (file));
    memset (f,0,sizeof (file));
    addsym (buf,&filenames,f);
    addlist (files,f);
    strcpy (f->name,buf);
    lex ();

    /* Fields */

    if (nt != T_OPENBRACE)
        error ("{' expected");
    lex ();

```

```

parsevarlist (&f->fields,&f->fieldnames);
if (nt != T_CLOSEBRACE)
    error ("'"' expected");
lex ();

    /* Index if any */

if (iskeyword (K_INDEX))
{
    if (nt != T_UNDEF)
        error ("Index field name expected");
    strcpy (f->index.name,buf);
    addsym (buf,&f->fieldnames,&f->index);
    lex ();
    f->index.type = V_STR;
    f->index.l = parseconst ();
}

    /* Is this file linked on other files? */

if (iskeyword (K_ON))
{
    if (nt != T_OPENBRACE)
        error ("'"{' expected");
    lex ();
    while (nt != T_CLOSEBRACE)
    {
        fi = alloc (sizeof (fileid));
        strcpy (fi->name,buf);
        lex ();
        addlist (f->on,fi);
    }
    lex ();
}

    /* Is this file a table? */

if (iskeyword (K_TABLE))
    f->table = parseconst ();
}

    /* Fix up all file pointers to point to correct files */

for (f=files; f; f=f->next)
    for (fi=f->on; fi; fi=fi->next)
    {
        strcpy (buf,fi->name);
    }

```

```

    fi->p = findsym (filenames);
    if (fi->p == 0)
        error ("%s' is not the name of a file",buf);
}

/* Output file definitions */

for (f=files; f; f=f->next)
{
    /* Structure definition */

    nl ();
    os ("typedef struct");
    nl ();
    os ("{");
    indent = 1;

    /* Links to next and previous records */

    if (!f->table)
    {
        nl ();
        os ("long next;");
        nl ();
        os ("long prev;");
        nl ();
    }

    /* Index field */

    if (f->index.l)
    {
        nl ();
        os ("char ");
        os (f->index.name);
        os ("["");
        oi (f->index.l);
        os ("];");
        nl ();
    }

    /* Files this one is linked on */

    for (fi=f->on; fi; fi=fi->next)
    {
        nl ();
        os ("long ");

```

```

    os (fi->p->name);
    os ("ptr;");
    nl ();
    os ("long ");
    os (fi->p->name);
    os ("next;");
    nl ();
    os ("long ");
    os (fi->p->name);
    os ("prev;");
    nl ();
}

    /* Files that are linked on this one */

for (f2=files; f2; f2=f2->next)
    for (fi=f2->on; fi; fi=fi->next)
        if (fi->p == f)
            {
                nl ();
                os ("long ");
                os (f2->name);
                os ("first;");
                nl ();
                os ("long ");
                os (f2->name);
                os ("last;");
                nl ();
                break;
            }

    /* Normal fields */

outvarlist (f->fields);

    /* End of structure definition */

indent = 0;
nl ();
os ("} ");
os (f->name);
os (";");
nl ();

    /* Blank record */

nl ();

```

```

os (f->name);
os (" blank");
os (f->name);
os (" =");
nl ();
os ("{");
indent = 1;

    /* Links to next and previous records */

if (!f->table)
{
    blankptr ();
    blankptr ();
    nl ();
}

    /* Index field */

if (f->index.l)
{
    outblank (&f->index);
    nl ();
}

    /* Files this one is linked on */

for (fi=f->on; fi; fi=fi->next)
{
    blankptr ();
    blankptr ();
    blankptr ();
    nl ();
}

    /* Files that are linked on this one */

for (f2=files; f2; f2=f2->next)
    for (fi=f2->on; fi; fi=fi->next)
        if (fi->p == f)
            {
                blankptr ();
                blankptr ();
                nl ();
                break;
            }

```

```

        /* Normal fields */

for (v=f->fields; v; v=v->next)
    outblank (v);

        /* End of blank record */

indent = 0;
nl ();
os ("};");
nl ();

        /* Record variable */

nl ();
os (f->name);
os (" ");
os (f->name);
os (";");

if (!f->table)
{

        /* File handle */

nl ();
os ("int ");
os (f->name);
os ("file;");

        /* Record pointer */

nl ();
os ("long ");
os (f->name);
os ("ptr;");

        /* Index file handle */

if (f->index.l)
{
    nl ();
    os ("int ");
    os (f->name);
    os ("indexfile;");
}
}

```

```

    nl ();
}

    /* Data in miscfile */

nl ();
os ("#define\t");
os (f->name);
os ("miscoff\t");
oi (miscoff);
nl ();

if (f->table)
    miscoff += reysize (f);
else
{
    miscoff += 3*sizeof (long);
    if (f->index.l)
        miscoff += sizeof (long);
}
}

    /* Procedures */

parsekeyword (K_PROCEDURES);

while (nt == T_UNDEF)
{
    /* Create procedure structure */

    p = alloc (sizeof (proc));
    addsym (buf,&procnames,p);
    addlist (procs,p);
    strcpy (p->name,buf);
    lex ();

    /* Get menu item name if present */

    p->menuitem = 0;
    if (nt == T_STR)
    {
        p->menuitem = alloc (strlen (buf) + 1);
        strcpy (p->menuitem,buf);
        lex ();
    }

    /* Output procedure heading */

```

```

nl ();
os ("void ");
os (p->name);
os (" (void)");
nl ();
os ("{");

    /* Indent one tab column within procedure */

indent = 1;

    /* Local variables */

parsevarlist (&locals,&localnames);

outvarlist (locals);
if (locals)
    nl ();

    /* Procedure code */

if (nt != T_OPENBRACE)
    error ("'{ expected");
lex ();
while (nt != T_CLOSEBRACE)
    parsestatement ();
lex ();

    /* Finish indented part */

indent = 0;

    /* Close brace after code */

nl ();
os ("}");

    /* Blank line before next procedure */

nl ();

    /* Free local variable list */

v = locals;
while (v)
{

```



```

    v2 = v->next;
    free (v);
    v = v2;
}
locals = 0;

    /* Free local variable symbol table */

    freesym (localnames);
    localnames = 0;
}

if (nt != T_EOF)
    error ("Procedure definition expected");

    /* Menu structure */

for (p=procs; p; p=p->next)
    if (p->menuitem)
        addoption (&mt,p->menuitem,p->name);

menuoutput (&mt,"menu");

    /* Main function */

nl ();
os ("main (void)");
nl ();
os ("{"");
indent = 1;
nl ();
os ("long r;");
nl ();

    /* Open or create files */

for (f=files; f; f=f->next)
    if (!f->table)
    {
        nl ();
        os (f->name);
        os ("file = opencreate (\\");
        os (f->name);
        os (".dat\\");
        if (f->index.l)
        {
            nl ();

```

```

        os (f->name);
        os ("indexfile = opencreate (\"");
        os (f->name);
        os (".idx\");");
    }
}
nl ();

/* Initialize misc file */

nl ();
os ("miscfile = open ("misc.dat",O_RDWR);");
nl ();
os ("if (miscfile < 0)");
nl ();
os ("{");
indent++;
nl ();
os ("miscfile = open ("misc.dat",O_CREAT | O_TRUNC | O_RDWR,0777);");
nl ();
os ("if (miscfile < 0)");
nl ();
os ("{");
indent++;
nl ();
os ("printf (\"Can't create file misc.dat\\n\");");
nl ();
os ("return 1;");
indent--;
nl ();
os ("}");
for (f=files; f; f=f->next)
{
    nl ();
    if (f->table)
    {
        os ("for (r=0; r!=");
        oi (f->table);
        os ("; r++)");
        indent++;
        nl ();
        os ("mwrite (&");
        os (f->name);
        os (",sizeof ");
        os (f->name);
        os ("");");
        indent--;
    }
}

```

```
    }
    else
        if (f->index.l)
            os ("mblank4 ()");
        else
            os ("mblank3 ()");
    }
    indent--;
    nl ();
    os ("}");
    nl ();

    nl ();
    os ("initvideo ()");
    nl ();
    os ("execmenu (menu,-1,-1,0,0)");
    nl ();
    os ("return 0;");
    indent = 0;
    nl ();
    os ("}");

    return 0;
}
```