SCULPTOR

INTRODUCTION TO SCULPTOR

AN INTRODUCTION TO SCULPTOR

Issued February 1986
Incorporating October 1986 Revision

ISBN 0 947770 02X

Published by
Microprocessor Developments Ltd.
3 Canfield Place
London NW6 3BT
England

CONTENTS

PAGE

CONTENTS (cont.)

PAGE

CHAPTER 1 INTRODUCING SCULPTOR CONCEPTS

This manual is intended to provide a basic explanation of the use of the SCULPTOR Program Development System. Reading this manual will enable you very quickly to develop the ability to set up data files and create programs, both for interactive screen use and to produce reports from the contents of your data files. When you have gained some experience of the basic functions, you can refer to the SCULPTOR Reference Manual for complete details of all the SCULPTOR features, including the more advanced techniques.

This chapter provides simple explanations of some of the basic concepts that underlie the SCULPTOR system. The chapter is divided into five sections:

1.1 FOURTH GENERATION PROGRAMMING

The term ''Fourth Generation'' is used to describe the major step forward in programming techniques which has been made in recent years. Third Generation programs were written in high level languages such as COBOL, BASIC or 'C', and then compiled or interpreted. The compiler would translate the high level language into machine code before attempting to run the program. The interpreter would decode the program instructions, step by step, as the program was run. Although the high level languages were powerful and a single command could represent a considerable number of machine code operations, it was still the responsibility of the programmer to create every last detail of the program structure. The programmer was, effectively, both the architect and the builder.

In Fourth Generation programming, an application program can be generated using a set of powerful development tools, each of which is specially designed for a particular part of the job. The programmer is still the architect, but the development aids take over the major part of the actual building work.

Many file management and database applications rely on fundamentally similar processes for storing, retrieving and manipulating information. A Fourth Generation Program Development System enables you to write complex and powerful programs by combining selected options and tailoring them to meet your own particular needs, without having to go right back to basics and laboriously code every step of the process.

The development tools that are included in the SCULPTOR system are:

- System for creating and maintaining Indexed Data Files
- Data Dictionary Descriptor program
- Screen Form Access and Update program
- Report Writer for printed output
- Menu program
- Automatic program generators
- Database Enquiry system

1.2 DATA STORAGE WITH SCULPTOR

The tremendous capacity of modern mass storage devices, coupled with the high speed of access and retrieval, make it essential to organise the storage of data as efficiently as possible. The sequence in which information is stored and the method of indexing can make a significant impact on the speed and efficiency of the retrieval and processing programs. To ensure that the explanations of data storage in SCULPTOR are understood without confusion, the following paragraphs describe some of the basic terms of reference used.

RECORDS

A record consists of a collection of pieces of information relating to a single entity, e.g. an employee, a stock item, a customer. For an employee, this information might include Name, Forenames, Date of Birth, Employee No., Date of Joining, Job Title, Grade, Rate of Pay, etc. Similarly, a stock item record would include information such as Item Description, Stock Code, Cost Price, Selling Price, Current Quantity, Re-order Level, Last Issue Date, Last Receipt Date, and so on.

FIELDS

Each of the pieces of information within a record is known as a Field. Different fields will contain different types of information, such as a date, a sum of money, or an item of text. For these different types of information, SCULPTOR uses different types of field, the types being alphanumeric, integer, date, money, and floating point fields.

FILES

A number of similar records grouped together forms a File. Thus a collection of Employee records forms an Employee File, and a collection of Stock records forms a Stock File. As files are also used to store other types of information besides data records, a file containing this type of data record is referred to as a Data File. Within a Data File, all the records will have exactly the same length assigned to them, regardless of the actual amount of data stored in each record. The record size is determined by adding together the lengths of all the fields that are contained in the record.

INDEXES AND KEYS

In order for the system to be able to access and retrieve a particular record from a data file, there must be an Index which points to the precise location of the required data record. To enable the Index to identify a record uniquely, each record must include a particular field (or a group of two or more fields) which contains some sort of unique identification, such as the stock code for a stock item record, or the employee number for an employee record. This field, or group of fields, is known as the Key.

One reason for some files having more than one field in the Key is simply to obtain a unique identification. For example, using the customer name for the Key in a customer file will not distinguish between two customers with the same name: adding a second field to the Key — perhaps the customer number — will provide the required unique identifier.

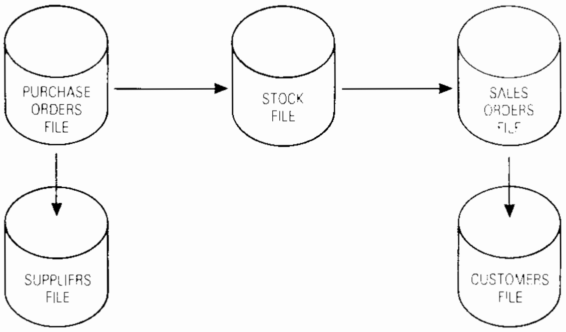
The method which the SCULPTOR system uses to index information is to create an Index File in association with each data file. This Index File contains only the Key Fields for the records in the data file, sorted into sequence and containing ''pointers'' that indicate the location of the full data record. This shorter Index File can be read by the system much more rapidly than the full data file, thus enabling the required data record to be extracted for processing more rapidly.

The actual structure of a SCULPTOR index is of a type known as a B-tree index, which is a particularly efficient indexing method, especially for large files. The SCULPTOR system automatically handles the maintenance of the index when new records are added to the file or old ones deleted. The only task for the programmer is to specify, when describing the record layout, which field (or fields) is to be used as the Key: all the rest is carried out by the system. A Data File which has been set up with an Index File is referred to as a Keyed File, the term being applied to the combination of both parts — the data and the index.

The choice of which fields to use as Key Fields within a record depends very much on the type of information being stored and the sort of processing which it is intended to perform on that information. Chapter 2 of this manual contains further details of choosing Key Fields.

A DATABASE

In a large or complex application, a number of data files may be interlinked to form a Database. The key field of the records in one file may also appear as a data field in another file, thus enabling cross-references between the two files during processing. The example below shows a simple interlinking of files:



	KEY FIELD	DATA FIELD ...			
STOCK RECORD:	STOCK CODE	DESCRIPTION	UNIT	COST PRICE	...
PURCHASE ORDER RECORD:	P/O NUMBER	SUPP CODE	STOCK CODE	QUANTITY	...
SUPPLIER RECORD:	SUPP CODE	SUPPLIER NAME	SUPPLIER ADDRESS	...	
SALES ORDER RECORD:	S/O NUMBER	CUST NO	STOCK CODE	QUANTITY	...
CUSTOMER RECORD:	CUST NO	CUSTOMER NAME	CUSTOMER ADDRESS	...	

When processing the Purchase Orders file, for example, the details of the stock item ordered can be picked up using the Stock Code as the key to the Stock File, and the supplier details can be similarly obtained from the Suppliers File using the Supp Code key.

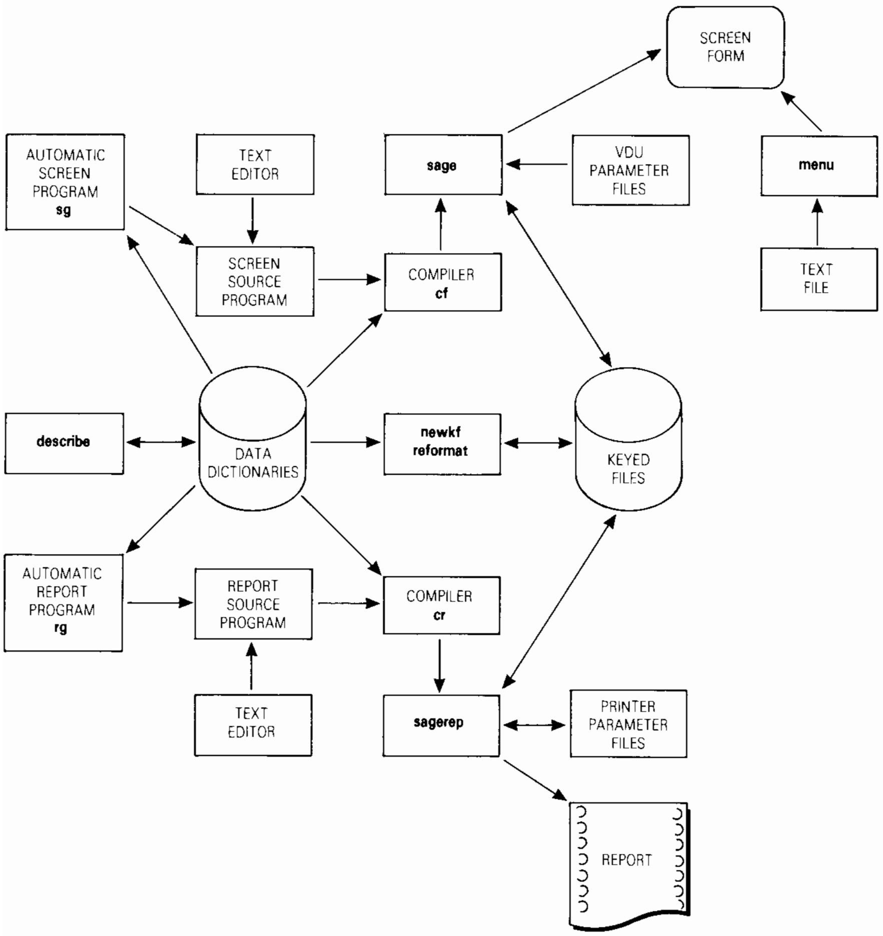
1.3 THE SCULPTOR PROGRAM SUITE

The SCULPTOR Program Development System consists of a number of programs which are designed to perform particular tasks in the building of a SCULPTOR application, and utility programs which are used when setting up the system and for maintenance and general support thereafter. The programs are as follows:

Program Name	Description
describe	The **describe** program is used to define the record layout for a data file. Both key fields and data fields are described, and the output of this program is referred to as a Data Dictionary.
newkf	The **newkf** program is used to create a new Keyed File from the record description specified in the data dictionary.
cf	**cf** is a compiler which reads your source language program (written using the SCULPTOR screen language commands), checks for errors, and produces a file of intermediate code for the **sage** interpreter to read at run time.
sage	The **sage** interpreter runs the interactive screen form programs compiled by **cf**.
cr	**cr** is a compiler which reads your source language program (written using the SCULPTOR report language commands), checks for errors, and produces a file of intermediate code for the **sagerep** interpreter to read at run time.
sagerep	The **sagerep** interpreter runs the batch processing and report programs compiled by **cr**.
menu	The **menu** program is a text file interpreter which produces a neatly formatted menu on the screen for access to the applications programs you have created.
query	The **query** program provides a simple method of displaying or printing information from SCULPTOR data files.

Program Name	Description
sg	The **sg** program is an automatic program generator for screen form programs. From the Data Dictionary output of the **describe** program, **sg** will create a screen form and generate an interactive program for it. An option within **sg** enables you to amend the standard details of the screen form and the program.
rg	The **rg** program is an automatic program generator for report programs. From the Data Dictionary output of the **describe** program, **rg** will create a report layout and generate the program to extract the information for the report from the data file. An option within **rg** enables you to amend the standard details of the report layout and the program.
sageform	The **sageform** program is used to provide a printed version of a screen form.
kfcheck, kfcopy, kfdet & kfri	The Keyed File Utilities provide facilities for checking the integrity, copying, displaying details, and rebuilding the index of Keyed Files.
reformat	The **reformat** program is used to reorganise a Keyed File when the record layout has been changed.
lcf	The **lcf** program is the means by which you can adjust the system for different language configurations, such as translations of standard text items and the format of dates.
setvdu/ decvdu	The VDU parameter file programs are used to set up and decode the parameters that describe the way in which your particular VDU terminal operates. (A set of standard parameter files is supplied with the SCULPTOR package.)
setprinter/ decprinter	The printer parameter file programs are used to set up and decode the parameters that describe the way in which your particular printer operates. (A set of standard parameter files is supplied with the SCULPTOR package.)

Diagram of the SCULPTOR Program Suite



Note: The diagram indicates only the major relationships between the modules, and is not intended to show all the interactions between the various programs.

1.4 DEFAULT VALUES

Throughout the SCULPTOR programs, there are many instances where the system expects a value to be input to define an item such as a field format, the gap between printed items, or the width or depth of the screen. If no value is input, the system assumes a standard value for the item. These standard values are known as default values. Considerable care has been exercised in selecting the default values so that, as often as possible, the most commonly required result can be achieved simply by doing nothing. Any of the default values, however, can be overwritten if a value other than the default value is required. The SCULPTOR Reference Manual gives details of the default values in the sections on the relevant programs and commands.

1.5 PROGRAMMING STANDARDS AND PRACTICES

The SCULPTOR system can be used to create a complex network of interlinked data files which are accessed, updated and processed by a large number of applications programs. Because of the numbers of both files and programs that can be involved in a single application, it is advisable to employ some basic programming standards and practices to ensure that errors are not caused simply by incompatible programming techniques.

An example of an advantageous adoption of a simple standard practice can be found in the naming of fields within data records (part of the **describe** program): to guard against possible duplication of field names, each field name can be preceded by a one or two letter abbreviation of the file name. Thus the order number field in the Purchase file record could be named p_ordno, and the order number field in the Sales file record could be named s_ordno. This simple technique goes a long way towards ensuring that each field name is unique.

Some examples of coding standards can be found in the sample programs given in Appendix A.

CHAPTER 2 KEYED FILES — How to Describe and Create them

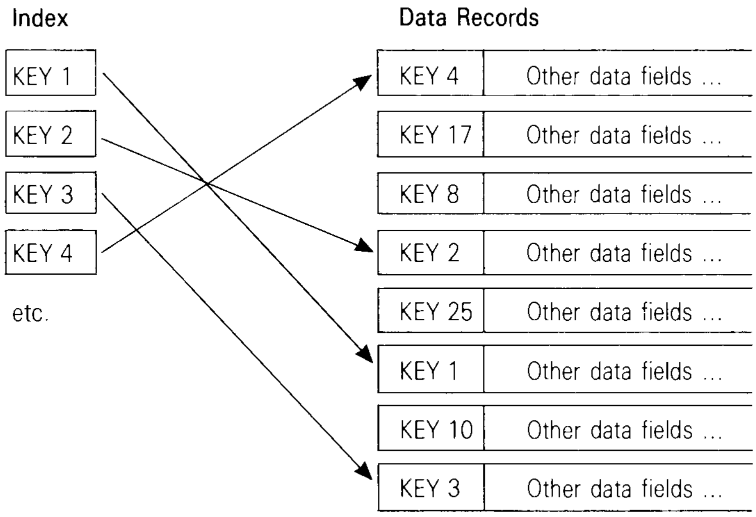
Keyed files (also referred to as keyfiles) form the database at the heart of each system created by SCULPTOR. This chapter describes the structure and organisation of data files and indexes which, together, form keyed files, and explains the use of the **describe** and **newkf** programs. The chapter is divided into five sections:

2.1 FILES AND FILE STRUCTURES

In any system that you create using SCULPTOR, information is stored in a database of index sequential files. Each of these index sequential files consists of a data file and an associated index. The index system used by SCULPTOR is known as a B-Tree index, which has the advantages that insertion, retrieval and deletion of records is very fast, and, since the index remains permanently sorted, no file re-organisation is necessary.

It is important to understand that the index is sorted and sequenced (logically, rather than physically) according to the field in each record which you designate as the Key Field. If you select the employee surname as the main key field for your personnel records, the index will be sorted, logically, into alphabetic order of employee surname. Reading the file and retrieving records in key field sequence provides for very rapid retrieval. However, if you attempt to read the same file without specifying the primary key (e.g. using the forename field only as the key), the benefits of the index sequential system are lost, and retrieval time can be much slower, particularly on a large file.

The diagram below indicates, very simply, the relationship between the records in the Index File and those in the Data File.



FILENAME CONVENTIONS

As you use the various SCULPTOR programs to create data files and indexes and to generate screen form programs and report programs, the system uses a simple method of naming to identify all the files that are associated with your original data file. It is worthwhile becoming familiar with these filename conventions even before the different activities are explained in detail.

The choice of name for your original data file is entirely your own (with the exception of a few SCULPTOR reserved words). However, if the application is intended to be portable, it is advisable to ensure that the various operating system restrictions are adhered to, for example — the MS DOS limit of eight characters with an optional three character extension.

File	Extension	Example
Data file	None	orders
Data dictionary	.d	orders.d
Index file	.k	orders.k
Screen program (source code)	.f	orders.f
Screen program (compiled)	.g	orders.g
Report program (source code)	.r	orders.r
Report program (compiled)	.q	orders.q
Menu program	.m	orders.m

FILE STRUCTURE

The structure of the data file is set up using the **describe** program to define the field sizes and types. This information is stored in the data dictionary for the file. However, the actual splitting of each record into the fields of the different types is a logical operation which is performed by the **sage** or **sagerep** interpreters at run time. Most of the keyfile utility programs are not aware of the internal structure of the file, with the exception of the location and size of the key within the record.

2.2 DATA DICTIONARIES

A Data Dictionary is a set of descriptions defining the type of data which is to be stored in a file. This is where the record structure of a file is separated logically into fields, and where the attributes of each field are specified. When a Data Dictionary has been created for a file, any programs that use the file will have access to the definitions that have been set up.

The first stage in developing a SCULPTOR application is to create the Data Dictionary for each data file. However, before starting work on your computer terminal, you should first decide on the basic data storage requirements for each record:

- What data items are to be stored in each record?
- How much space is required for each field?
- What type of information is to be stored (e.g. text, date, money)?
- Which field (or fields) will be used as the Key?

This last point is very important. The key used must always produce a unique index reference to each record, so the possibility of two records with the same key must be completely eliminated. If the first data field that you consider for the Key Field does not achieve this, then it will be necessary to specify additional fields to form the Key. (Refer to Section 2.3 of this manual for examples of choosing Key Fields.)

The other point about the selection of the Key is that the logical ordering of the index sequential file will be in Key sequence. You should therefore be sure that the key you select is going to provide the required sequence for retrieval and processing of records within the file.

When you have decided on the basic elements of your file design, use the SCULPTOR program called **describe** to create a Data Dictionary. Section 2.4 of this manual gives details of using the **describe** program.

2.3 CHOOSING KEY FIELDS

There are two purposes in choosing fields from a data record to form the Key for the record:

1. Providing a unique identifier for the record.
2. Sorting the file into the required sequence.

If you specify more than one field to form the key, the hierarchy of the sorting sequence will correspond to the order in which you specify the key fields. Thus Key Field 1 gives the main sequence; Key Field 2 gives the subsidiary sequence within the main sequence, and so on.

Consider the following examples of Key Fields selected for a file to store customer records:

Key Fields	**Comments**
forename, surname	This may provide a unique key, but the order of the Key Fields will cause the file to be sorted into Forename sequence.
custno, surname	Assuming that the Customer Number is unique for each customer, the Surname field need not be included in the Key as it is not serving either of the two required purposes.
custref	If a suitable customer code system is chosen (e.g. an abbreviation of the surname), this is a simple method of achieving fast direct access and a sensible file sequence.
surname, custno	This provides file sequencing in surname order, which is desirable for reports. If fast direct access is required by ''custno'' alone, a second index file must be built with an inverted key (i.e. custno, surname).

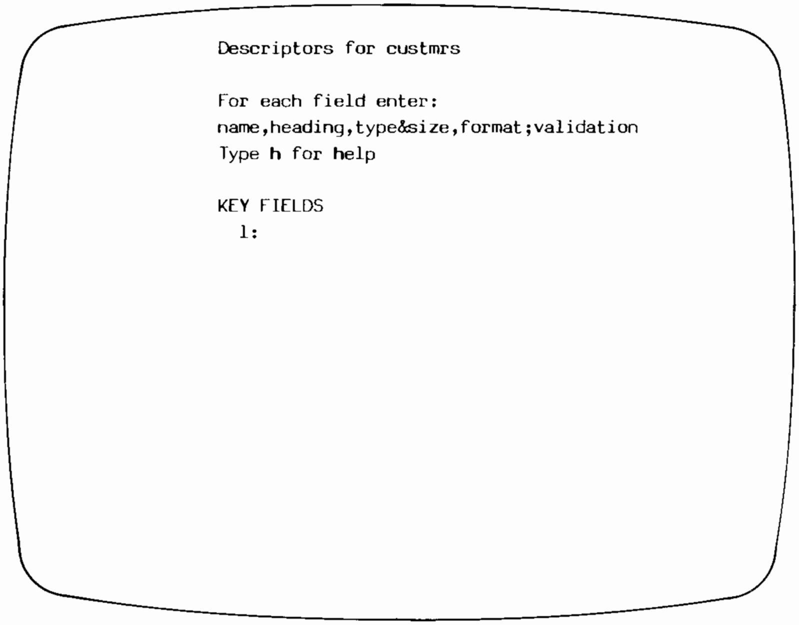
Any type of field can be used as a key field, but it is advisable to avoid using floating point fields and fields which may contain negative numbers, as these can have undesirable effects on file sequencing.

2.4 THE describe PROGRAM

The first stage in the creation of a SCULPTOR file is the definition of the record structure using the SCULPTOR program called **describe**.

ACCESS

On the SCULPTOR Development Menu, select option 1. The system waits for you to enter the name of the file for which you want to create a Data Dictionary. Alternatively, type **describe <filename>** to call the program from your directory. When you enter the name of your data file (in the example below, the filename is ''custmrs'') the system displays the following text:



```
Descriptors for custmrs

For each field enter:
name,heading,type&size,format;validation
Type h for help

KEY FIELDS
  1:
```

The cursor waits for you to type in the field description for Key Field 1. When you finish the Key Field 1 description and press return (or enter), the system displays "2:" on the next line and waits for the description for the second key field. If you do not want a second key field, simply press return to move on to describing the data fields.

Each line of description consists of entries for the following items, typed one after the other, but separated by commas:

name	The field name.
heading	The heading to be displayed on the screen or in reports.
type&size	The type of data to be stored and the size of the field.
format	Special instructions for the format of the field when displayed or printed in reports.
validation	Specifying limitations as to the permitted values that can be stored in this field.

Each of these parts of the description is explained briefly in the following paragraphs.

FIELD NAMES

Each field must be given a reference name which will be used to access the field by all programs that read, process or update the file. It is recommended most strongly that field names should be selected which are concise, meaningful and unlikely to be duplicated in other files within the database. A useful field naming practice to adopt is to prefix each name with a one or two letter abbreviation of the file name. File names can contain alphabetic characters, the underscore character (_), and the numerals 0 to 9. The first character must not be a numeral, and all other punctuation and special characters are prohibited. For all practical applications, there is no limit to the length of the field name. Examples are as follows:

su_name	SUPPLIERS file, the field to contain the supplier name.
tr_date	TRANSACTION file, the field to contain the transaction date.
ordquant	the field to contain the order quantity.

Pressing return instead of entering a field name will end entry of Key Field descriptions or Data Field descriptions, whichever was being entered at the time. After entering the field name, type a comma and then enter the heading.

FIELD HEADINGS

The field heading you enter here will be stored by SCULPTOR as the default heading for the field, to be used when the field is displayed on the screen or printed in reports. When you are generating screen form programs or report programs, you will be able to override this default heading, if required. There are no restrictions as to the characters used for the heading, but, if it is to contain any punctuation characters, the entire heading should be enclosed in quotation marks.

Entering a single space followed by a comma will produce a blank heading. If you enter nothing at all for the heading (i.e. two consecutive commas), the field name will be used as the heading. (Remember that the entire field name will be used as the heading — including the file prefix, if used.)

Examples of field names and headings:

su_name,Supplier Name,	
tr_date,TRANSACTION DATE,	
o_quant, ,	(no heading)
quantity,,	(field name ''quantity'' used as heading)
c_freq,"LOW, MED or HIGH",	(heading contains a comma)

FIELD TYPE AND SIZE

This entry is used to describe the type of data to be stored in the field and the storage size. For numeric fields, the size does not correspond to the number of digits, but indicates a range. The permitted letters (indicating type) and numbers (indicating size) are as follows:

a*n* = alphanumeric text	*n* = length of text, up to 255 chs.	
i*n* = integer (whole number)	*n* = 1	0 to 255
	n = 2	-32767 to 32767
	n = 4	Refer to Section 2.5 of the SCULPTOR Reference Manual for exact details of field size limitations.
d*n* = date	*n* = 4	
m*n* = money	*n* = 4	
	n = 8	
r*n* = real (i.e. decimal, or floating point)	*n* = 8	

Examples of field names, headings, type and size:

```
su_name,Supplier Name,a30
tr_date,TRANSACTION DATE,d4
o_value,Value,m4
quantity,,i2
```

The Format and Validation entries are optional, so, if you do not want to enter any value for these items, you can just press return to move on to the description of the next field.

FORMAT

The format description enables you to provide special instructions for the display of the field on the screen, or the way in which it will be printed in reports. If you do not make any entry here, SCULPTOR will use suitable defaults for the type and size of field. Some of the possible entries are shown below: refer to Chapter 2 of the SCULPTOR Reference Manual for further details.

Alphanumeric fields:	u = change input to upper case l = change input to lower case e = do not display input (e.g. for passwords)
Numeric fields:	# = digits, leading zeroes suppressed 0 = digits with leading zeroes * = digits with leading asterisks
Date fields:	dd/mm/yyyy (for example — used to alter the standard date format)

Examples:

su_name,Supplier Name,a30,u	name will appear in upper case
tr_date,TRANSACTION DATE,d4,"dd mm yy"	spaces for date separators
o_value,Value,m4, #####.##	no leading zeroes

If validation values are to be specified, the format should be terminated with a semi-colon. Otherwise, you can just press return to move on to the next field description.

VALIDATION

When you are running your screen form program, the data that you input is automatically validated by the system to ensure that the data is the correct type and size for the field. For example, a number containing a decimal point would be rejected as input for an integer field. Further validation can be specified by adding a list of the acceptable values to the field description. This should be done with care, as restrictions that are imposed at this stage can not be overridden later, when running the program.

Both individual values and ranges of values can be specified. Ranges are defined by typing the lower value followed by a hyphen and the upper value, such as 0-9. Each value or range of values is separated by a comma. If ''no input'' is acceptable to your program specification, this should be included in the validation list by typing two successive commas. The validation list is preceded by a semi-colon.

Examples of field descriptions with validation lists:

```
c_freq,"LOW, MED or HIGH",a1,u;L,M,H
m_chno,"CHASSIS NO.",i2,00000;,00100-29999,40000-69999
```

The first example will only accept entries of L, M or H (although entries of l, m or h would be accepted and changed into upper case because of the Format descriptor — u). The second example will accept ''no input'', or a number in one of the specified ranges.

Chapter 2 of the SCULPTOR Reference Manual gives further details of using validation lists.

When you have typed your validation list for a field, press return to move on to the next field description.

COMPLETING THE RECORD DESCRIPTION

When you have no further fields to describe, press the return key to complete the description. The system displays a line of options:

List, Change, Delete, Insert, Abandon, Save, Help

To select an option, type the initial letter of the option and press return. If you are satisfied with the descriptions you have entered, type S for the Save option to store the record description. The other options are explained in Chapter 2 of the SCULPTOR Reference Manual.

CHANGING AN EXISTING DESCRIPTION

When you access the **describe** program, entering the name of an existing descriptor file enables you to alter any of the field descriptions for that record type. (Incidentally, when entering the filename you need not type in the **.d** extension.) The system then displays the list of options given above. You can then select the required option and make the necessary alterations to the field descriptions.

WARNING: If fields are inserted or deleted or if field sizes are altered, the existing keyed file must be reformatted using the **reformat** program, and all the **sage** and **sagerep** programs that access the file must be re-compiled. This is essential to avoid file corruption. Complete details of the **reformat** program are given in Chapter 5 of the SCULPTOR Reference Manual.

MONEY FIELDS AND DATE FIELDS

Money fields are stored in the lower currency unit (e.g. pence for sterling, or cents for US $), which helps to avoid rounding errors. Money values which are input in the screen form program are validated to ensure that they are either integer values of the higher currency unit, or that they contain exactly two digits following the decimal point. For example:

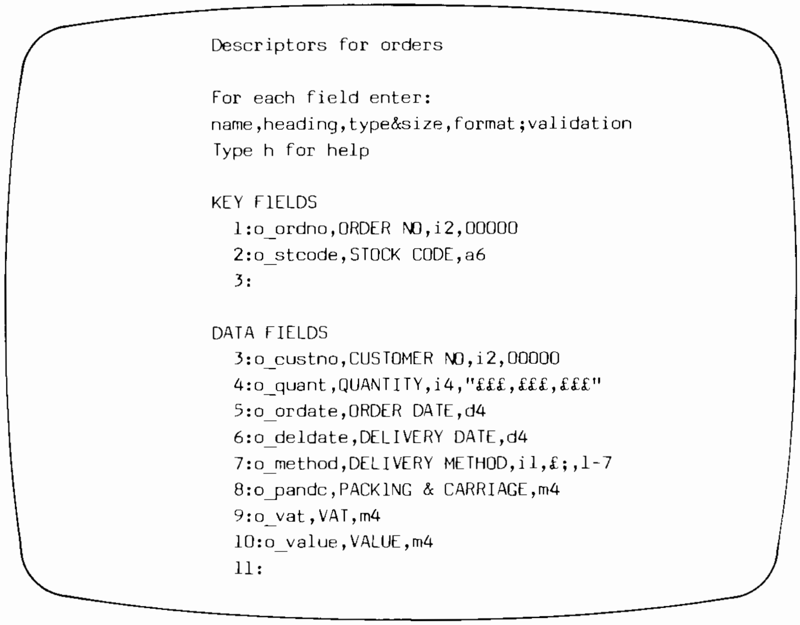
INPUT:	STORED AS:
125	12500
12.5	Invalid
12.50	1250
12.508	Invalid

If, in a program, you are assigning a value to a money field (as opposed to inputting a value on the screen form), you must remember to use the lower currency unit. E.g. to assign a value of $12.50, your statement must use the value 1250.

Date fields are stored internally as a day number, the sequence starting with day 1 being 1/1/0001. The format for the input and display of dates can be set using the Language Configuration program **lcf**. Most separators are permitted, and either two or four digits can be used for the year. If, in a program, you are assigning a value to a date field (as opposed to inputting a date on a screen form), you must remember to use the semi-colon separator, e.g. 31;3;86. If you assigned a value of 31/3/86, the value would be interpreted as an arithmetical expression, calculated as 31 divided by 3 divided by 86.

EXAMPLE

The example below shows the screen contents when the field descriptors have been entered for a simple sales orders file:



```
Descriptors for orders

For each field enter:
name,heading,type&size,format;validation
Type h for help

KEY FIELDS
  1:o_ordno,ORDER NO,i2,00000
  2:o_stcode,STOCK CODE,a6
  3:

DATA FIELDS
  3:o_custno,CUSTOMER NO,i2,00000
  4:o_quant,QUANTITY,i4,"£££,£££,£££"
  5:o_ordate,ORDER DATE,d4
  6:o_deldate,DELIVERY DATE,d4
  7:o_method,DELIVERY METHOD,i1,£;,1-7
  8:o_pandc,PACKING & CARRIAGE,m4
  9:o_vat,VAT,m4
  10:o_value,VALUE,m4
  11:
```

It is interesting to note that the sterling pound sign has replaced the # character in the numeric formats in the above example. In the ASCII character set, these two characters have the same value. Some terminals and printers display a pound sign, others a # character.

2.5 THE newkf PROGRAM

When the Data Dictionary has been created using the **describe** program, the second stage in the process of creating a SCULPTOR file is to create the new, empty key file. This is achieved using the **newkf** program.

To call up the **newkf** program, simply type newkf followed by a space and the name of the file you want to create. For example:

```
newkf custmrs
```

The Data Dictionary for the ''custmrs'' file must already exist for **newkf** to read the record size and key field location. When **newkf** has read this information and created the key file, the system displays the message:

```
custmrs created
```

WARNING: **newkf** creates an empty file. If the named file already exists, it is recreated as a new, empty file and the existing data is destroyed.

More than one key file can be created from one **newkf** command statement by listing all the required filenames, separated by spaces. For example:

```
newkf afile bfile cfile
```

Two additional features exist for the **newkf** program, selected by typing either -i or -rn after **newkf** and before the filename. The -i option creates only an index file with no data records. The -rn option, which is only normally used with the OS9 operating system, creates a file with space for the number of records specified by the number n, for example:

```
newkf -r2500 customers
```

Further details of these options are given in Section 5.1 of the SCULPTOR Reference Manual.

When you have created the key files for your SCULPTOR application, you are then ready to start writing screen form programs to input and amend information for the files, and report programs to produce printed output from the filed information.

CHAPTER 3 AUTOMATIC PROGRAM GENERATORS

When you have defined the record layout and created your keyfile using the **describe** and **newkf** programs, the next stage in building your SCULPTOR application is to create two types of program: a screen form program to enable you to input, amend and delete data records in the keyfile, and a report program to produce printed reports from the data.

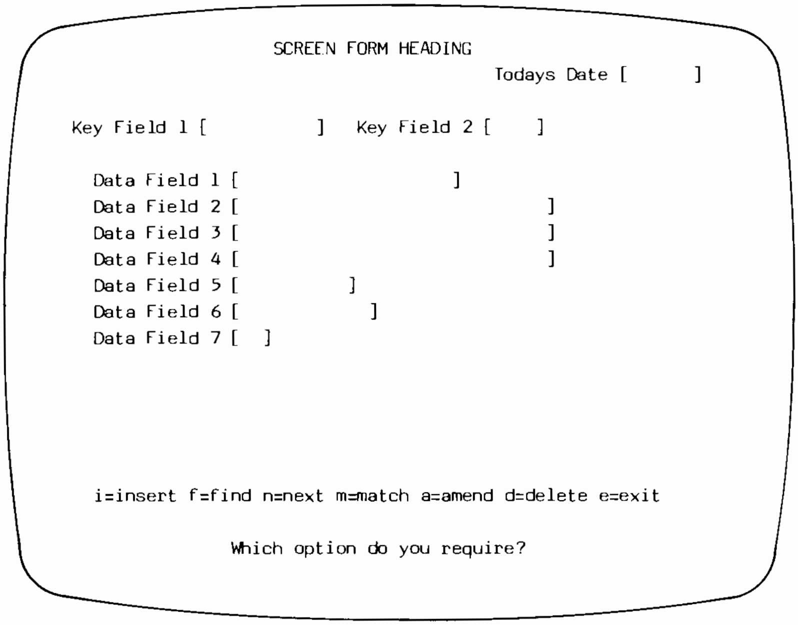
There are two methods of generating these programs. The first method is to use the Automatic Program Generators **sg** and **rg** to create standard programs. The second method is to use the SCULPTOR screen and report languages to write your own programs. This chapter explains the first method; the second method is described in Chapters 4 and 5.

This chapter is divided into two sections:

3.1 AUTOMATIC SCREEN FORM PROGRAM GENERATION (sg)

The utility program **sg** is used to create a standard screen form program. Using the Data Dictionary created by the **describe** program, **sg** formats the fields from the data record into boxes on a screen form, and prepares a standard list of options for the input, amendment or deletion of records from the file. The way in which the fields are arranged on the screen will depend on the number and size of the fields in the record. If there are more fields than can be included on the screen, **sg** will display a message to that effect and the program will terminate. The **sg -a** option should then be used to select the fields required to be displayed (see page 3-3).

The example below shows a typical screen form layout that **sg** would produce for a simple file having two key fields and seven data fields:



```
                          SCREEN FORM HEADING
                                              Todays Date [      ]

Key Field 1 [          ]  Key Field 2 [    ]

  Data Field 1 [                   ]
  Data Field 2 [                           ]
  Data Field 3 [                           ]
  Data Field 4 [                           ]
  Data Field 5 [          ]
  Data Field 6 [            ]
  Data Field 7 [  ]




  i=insert f=find n=next m=match a=amend d=delete e=exit

                Which option do you require?
```

ACCESS

To create your standard screen form program, simply type **sg** followed by the name of the data file you want to use. For example:

```
        sg custmrs
```

Note that the data dictionary for the ''custmrs'' file must already have been defined using the **describe** program. **sg** then obtains the key field and data field details from **custmrs.d** and creates a new file, **custmrs.f**, containing the source code of the screen form program. This file is automatically compiled to produce the object code file — **custmrs.g**.

To run your screen form program, type **sage** followed by the filename, for example:

```
        sage custmrs
```

ALTERNATIVE PROGRAMS

If you want to create a screen form program that differs from the standard, you can use the **sg -a** option, for example:

```
        sg -a custmrs
```

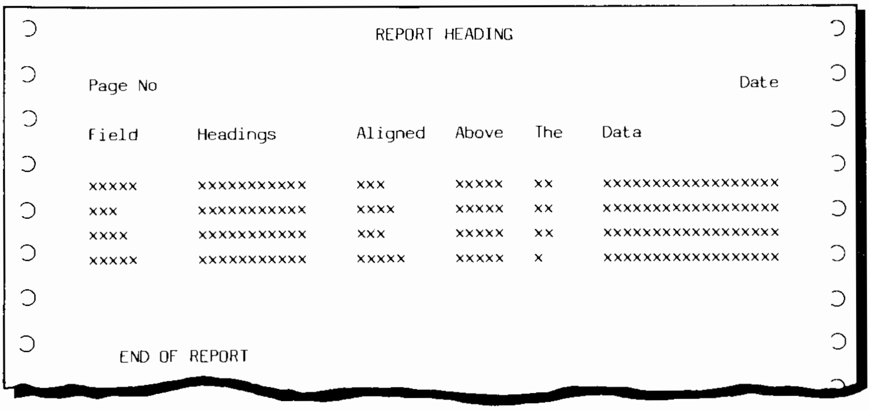
This option enables you to change the program name, the screen title, the field headings and the output formats from the defaults that would be provided by **sg**. It also allows you to select particular fields for the screen form in instances when **sg** has been unable to create a screen form with all the fields on one screen. When you use this option, the system displays a prompt screen on which you can select and enter the amendments that you require.

Note: You can also make amendments to the standard program produced by **sg**. This is done by calling up the source code file (e.g. **custmrs.f**) in your text editor and amending the actual program statements. When you have altered your program using this method, you will also have to recompile the program using the compiler **cf**. To attempt this method, however, you should already be familiar with the writing of a **sage** program, as described in Chapter 4.

3.2 AUTOMATIC REPORT PROGRAM GENERATION (rg)

The utility program **rg** is used to create a standard report program. Using the Data Dictionary created by the **describe** program, **rg** formats the fields from the data record into an appropriate set of headings and columns for a printed report. The actual arrangement of the fields in the report will depend on the number and size of the fields in the record. If there are more fields than can be accommodated on one line of the report, you can use the **rg -a** option described on page 3-5 to select the fields to be included in the report layout. This option is recommended for most applications, unless the file consists of records that contain only a few, short fields.

The example below shows a typical report layout that **rg** would produce for a simple file containing six fields:



```
                              REPORT HEADING

     Page No                                                    Date

     Field      Headings      Aligned   Above   The    Data

     xxxxx      xxxxxxxxxxx   xxx       xxxxx   xx     xxxxxxxxxxxxxxxxxx
     xxx        xxxxxxxxxxx   xxxx      xxxxx   xx     xxxxxxxxxxxxxxxxxx
     xxxx       xxxxxxxxxxx   xxx       xxxxx   xx     xxxxxxxxxxxxxxxxxx
     xxxxx      xxxxxxxxxxx   xxxxx     xxxxx   x      xxxxxxxxxxxxxxxxxx


        END OF REPORT
```

ACCESS

To create your standard report program, simply type **rg** followed by the name of the data file you want to use. For example:

```
rg orders
```

Note that the data dictionary for the Orders file must already have been defined using the **describe** program. **rg** then obtains the field details and field headings from the file **orders.d** and creates a new file, **orders.r**, containing the source code of the report program. This file is automatically compiled to produce the object code file — **orders.q**.

To run your report program, type **sagerep** followed by the filename, for for example:

```
sagerep orders
```

ALTERNATIVE PROGRAMS

If you want to create a report program that differs from the standard, you can use the **rg -a** option, for example:

```
rg -a orders
```

This option enables you to change the program name, the report title, the field headings and the output formats from the defaults that would be provided by **rg**. It also allows you to select particular fields for the report in instances where **rg** has not been able to accommodate all the fields on one line. Other amendments you can make are suppressing the display of repeated key values, and producing totals for numeric fields. When you use this option, the system displays a prompt screen on which you can select and enter the amendments that you require.

Note: You can also make amendments to the standard report program produced by **rg**. This is done by by calling up the source code file (e.g. **orders.r**) in your text editor and amending the actual program statements. When you have altered your program using this method, you will also have to recompile the program using the compiler **cr**. To attempt this method, however, you should already be familiar with the writing of a **sagerep** program, as described in Chapter 5.

CHAPTER 4 SCREEN FORM PROGRAMS (sage)

Screen form programs are used to input data to keyfiles, the data being entered in specially formatted boxes on the screen that correspond to the fields in the data record. Subsequently, the screen form programs are also used to insert or delete records and to update or amend the existing information.

A standard screen form program can be created using the **sg** program, as described in Section 3.1. These standard programs are entirely satisfactory for straightforward applications, but they do have limitations, such as the restriction of only one data file. If you want to create a more complex program which can read several data files and perform more specialised file operations, it is necessary to write your own source code program using the SCULPTOR screen language.

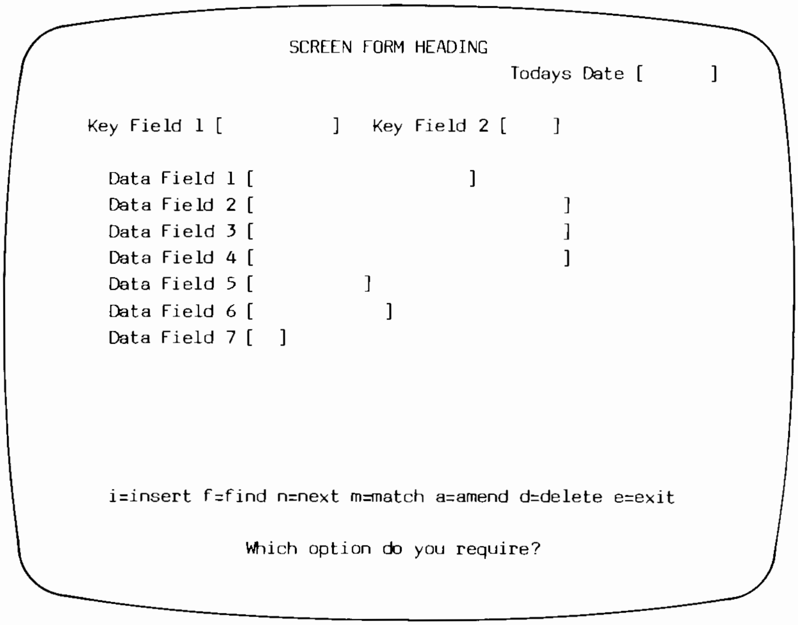
The source code program is written using whichever text editor is available on your system. When the program is complete, you use the compiler program **cf** to read your source code text file and create the object code program for use at run time.

This chapter describes the structure of **sage** programs and explains the use of a selection of the SCULPTOR screen language commands. Complete details of writing **sage** programs can be found in Chapter 3 of the SCULPTOR Reference Manual.

This chapter is divided into six sections:

4.1 SCREEN FORM PROGRAM STRUCTURE

The structure of a screen form program relates to the layout of the final product — the screen form. The example below is a typical screen form, as produced by the program **sg**:



```
                          SCREEN FORM HEADING
                                              Todays Date [        ]

     Key Field 1 [          ]  Key Field 2 [    ]

       Data Field 1 [                   ]
       Data Field 2 [                            ]
       Data Field 3 [                            ]
       Data Field 4 [                            ]
       Data Field 5 [          ]
       Data Field 6 [            ]
       Data Field 7 [  ]




       i=insert f=find n=next m=match a=amend d=delete e=exit

                    Which option do you require?
```

At the top of the screen is the Screen Form Heading, or title line. This is followed by the date field, and then a field heading and space for the appropriate amount of data for each of the key fields and data fields, the space being indicated by the delimiting characters: [   ]. At the foot of the screen is the line specifying the list of options for the file operations that can be performed, and the prompt asking for an option to be selected.

To produce such a screen form, the **sage** program must perform the following tasks:

- Define a title for the program.
- Identify all the files that will be accessed by the program, and any temporary fields that are required, such as the system date field.
- List all the fields that are to be displayed on the screen and specify their position on the screen by row and column number.
- Specify the options for the list at the foot of the screen, and, for each option, define the processing that should take place when the option is selected.

The program structure that enables these tasks to be achieved consists of different types of program line, which are identified by the initial character on the line:

The title line
The first line in the program is the title line, irrespective of the initial character.

Declarations
Lines that start with an exclamation mark (!) are declarations. These are statements that identify the files that are to be read, specify temporary fields, define different box delimiting characters, and perform other initialising tasks.

Box Definitions
Lines that start with a plus sign (+) are box definitions. These are statements that specify the position on the screen of the boxes for each of the fields that are to be displayed.

Options
Lines that start with an asterisk (*) are options. Each option specifies the code and option name, and is followed by the program statements that specify the processing that is to take place when that particular option is selected.

Comments
Comment lines can be included anywhere in the program by starting the line with a period: these lines are ignored by the program compiler.

Program Statements
Lines starting with any other character are program statements. If the first word on the line is not recognised as one of the SCULPTOR screen language words or a field name, it is taken to be a line label. Multiple statements can be placed on one line by separating them with colons. To continue a statement onto a second line, the first line should be terminated with a backslash (\).

A typical **sage** program structure groups these different line types together as follows:

Screen Form Title line	
!declaration !declaration !declaration	declarations, as required
+box definition +box definition . . +box definition	box definitions for each field to be displayed on the screen form
*option program statements *option program statements *option program statements	option names, and their associated program statements, for each option to be included at the foot of the screen form
subroutines	subroutines are identified by line labels, and can appear anywhere in the program

4.2 sage DECLARATIONS

After the screen form title line, the first section of a **sage** program contains one or more declarations. These lines, commencing with an exclamation mark, are used to list the files which are to be accessed by the program, and to set up other initial features of the program. The declarations that are available in **sage** are listed below. The full explanations of the declarations can be found in Section 3.9 of the SCULPTOR Reference Manual.

FILE DECLARATIONS

!file Declare a file which is initially open. A separate declaration is required for each file that is to be accessed. A file can also be declared using a file identifier and a pathname (the pathname is omitted when the file identifier is the filename). The file identifier is then used to refer to the file in subsequent file access commands. For example:

```
!file orders
!file a1 /usr/acc/accounts
```

!cfile Declare a file which is initially closed. This is frequently used to increase the number of files declared above the operating system limit. For example:

```
!cfile rates
```

!record Declare an alternative record layout for a file. The file must already have been declared, and the alternative layout must have been defined using the **describe** program. For example:

```
!record orders orders1
```

SCREEN DECLARATIONS

!box Define box delimiters to be used on the screen instead of the standard ones, which are normally [ ]. For example:

```
!box < >
```

!depth Define the number of lines required for the screen depth (the default being 24). For example:

```
!depth 20
```

!width Define the number of columns required for the screen width (the default being 80). For example:

```
!width 132
```

OTHER DECLARATIONS

!temp Declare a temporary field for use within the program. Fields that you declare here can be used in the program in the same way as ordinary keyfile fields. As in the **describe** program, fields must have a name, can have a heading, and must have a type and size. For example:

```
!temp total,Total,m4
```

These items are defined in the same way as for field descriptions in the **describe** program: refer to Section 2.4 of this manual for details. (Temporary fields cannot have automatic validation lists.) Some special temporary fields can also be declared: these are described in Section 3.9 of the SCULPTOR Reference Manual.

!scroll Define an area on the screen where data is to be displayed in columns under a heading line. The two numbers after the declaration define the line number of the heading line, and the number of rows of boxes required in each column. For example:

```
!scroll 8,12
```

(Twelve rows of boxes under the headings in line 8.)

For a simple application, you will not require to declare closed files, alternative record layouts or scroll areas, and the screen declarations can be left as the default values. An example of the declarations for a simple program could be as follows:

```
ORDER ENTRY PROGRAM

!file orders
!file stock
!file custmrs

!temp date,Today's Date,d4
!temp total,Total,m4
```

Note that blank lines can be included in the program to improve clarity: like comment lines, they are ignored by the compiler.

4.3 SCREEN FORM DESIGN

The design of the screen form is achieved using the Box Definitions. All fields for which values are to be input or displayed in the program must be given a box definition line starting with a plus sign. The screen boxes can be defined in any order. However, because input and display of a range of fields is carried out in the order in which the boxes were defined, it is usual to write the box definitions in a logical order of screen position.

A box definition consists of the following elements:

+ field name, field heading, row, column, format

The field heading and the format are optional, but the name, row and column must be specified.

When designing your screen form, do not position boxes on the top line or the bottom three lines of the screen. These lines are required for the screen form title, the menu line, the prompt and the message area.

Field Name

The name must be that of a field from one of the declared files, or of a declared temporary field. Terminate the name with a comma.

Field Heading

Type in the heading to be displayed to the left of the box, terminating with a comma. If you make no entry for the heading (i.e. two consecutive commas), the field heading from the data dictionary for the file is used. If you want a blank heading, leave a single space between the commas.

Row and Column Numbers

The position specified by the row and column numbers is the position of the first character of the data in the field, so you must remember to leave space for the field heading and the box delimiter.

Format

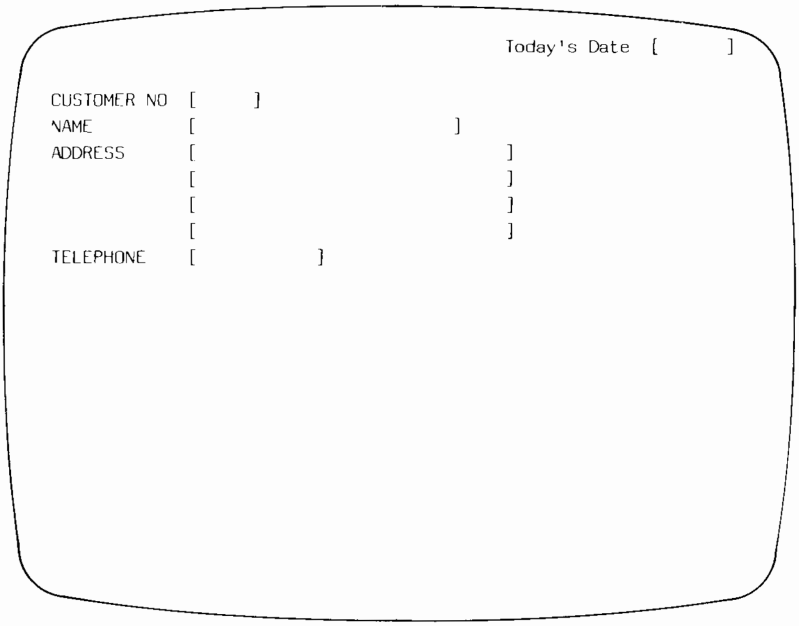
If no entry is made for the format, the format from the data dictionary is used. If you want a format different from the data dictionary one, type a comma after the column number and define the required format in the same way as in the **describe** program (refer to Section 2.4 of this manual).

For each box definition, the system will create a screen box bounded by the box delimiting characters. These are normally [ ] , but can be altered with a !box declaration, or by changing the default value in the utility program **lcf**. For alphanumeric fields, the box width equals the field width. For other field types, the field width depends on the format, which is either specified in the box definition, or defaulted to the format from the data dictionary.

Box Definition Examples

```
+date,Today's Date,2,70
+cu_code,CUSTOMER NO,4,20
+cu_name,,5,20
+cu_addr1,,6,20
+cu_addr2, ,7,20
+cu_addr3, ,8,20
+cu_addr4, ,9,20
+cu_tel,,10,20
```

These examples produce screen boxes positioned as in the illustration below.



```
                                        Today's Date  [        ]

CUSTOMER NO  [      ]
NAME         [                         ]
ADDRESS      [                              ]
             [                              ]
             [                              ]
             [                              ]
TELEPHONE    [            ]
```

4.4 sage PROGRAM OPTIONS

When the declarations and box definitions have been completed, the remaining part of a **sage** program consists of the options for the menu line at the foot of the screen. Each option for the operations to be performed on the data files (such as inserting, amending or deleting records) is identified by an option title line, which is followed by the program statements for that option.

The option title line starts with an asterisk and has the following format:

***<code> = <description>**

For example:

***f=find**

The code can be any one or two printable characters. The descriptions should be kept short and the number of options fairly small as the options are all merged into a single menu line at the foot of the screen. The prompt line that appears beneath the menu is produced automatically by the system. The menu line and prompt for a typical program might appear as follows:

```
i=insert a=amend d=delete f=find n=next e=exit
           Which option do you require?
```

The program options part of such a program would have six option sections each starting with the relevant option title line, e.g. *i=insert, *a=amend, etc. When one of the options is selected during the running of the program, control is passed to the commands in the section following that option's title line. Processing of those commands continues until it is terminated in one of the following ways:

- An **end** statement returns control to the option prompt.
- An **exit** statement terminates the program completely.
- An error condition causes a message to be displayed: control is returned to the option prompt.
- The CANCEL key is used to cancel input for an option and return to the option prompt.

The commands that can be used in the program statements are described in the following pages.

There are more than forty commands in the SCULPTOR screen language. For a complete list of the commands and a full explanation of each command, refer to Section 3.10 of the SCULPTOR Reference Manual. In the following pages, the different categories of command are explained briefly with selected examples of particular commands.

KEYFILE CONTROL COMMANDS

These commands are used to control the use of the declared keyfiles. The available commands include:

open, close, rewind, unlock

For example, if the orders file was declared as a closed file (using the **!cfile** declaration), the program statements would need to include a command to open the orders file before any attempt is made to access it. The syntax for such a command is simply:

open orders

KEYFILE ACCESS COMMANDS

When a keyfile is declared (see Section 4.2 of this manual), the system sets up a record buffer for the file. The access commands are used to search the keyfile for a particular record and to read the data into the record buffer. The available commands include:

read, find, next, prev, match, readkey, nextkey, testkey, check

The **read** and **find** commands are used for direct access to a file. The filename must be specified and there is an option to specify a key other than the file's natural keyfields. The **read** command transfers a record to the record buffer only if there is an exact match with the key field; the **find** command transfers a record if there is a partial match.

The **next**, **prev** and **match** commands are normally used after a direct access command to transfer to the record buffer the record whcse key is next higher (**next**), next lower (**prev**) or the same as (**match**) the key used by the last direct access command. That is, the current value of the index pointer in the index sequential file.

The **readkey**, **nextkey** and **testkey** commands search the index part of the file, but do not transfer a record to the buffer. Their advantage is that these operations are successful even if the associated data record is locked.

The **check** command is used to check that a record has been read from the specified file and not written back or cleared.

An example of a **read** command with an alternative key specified:

```
read custmrs key=ordno
```

KEYFILE UPDATE COMMANDS

These commands are used to update the keyfile data by inserting new records, writing amended information back to existing records and deleting records. The commands are:

insert, write, delete

For the **delete** command, the key field value currently in the record buffer must correspond to a record in the file (otherwise there is no record to delete). For the **insert** command, the reverse is true: the current key field value must be a new value that does not already exist in the keyfile.

For these commands, it is only necessary to specify the filename: the actual record is determined by the current contents of the record buffer. For example:

```
insert custmrs
```

SCREEN FORM COMMANDS

These commands are used to control the input and display of data on the screen form. The commands include:

input, display, clear, preserve, newform, highlight,

For the **input** and **display** commands, the syntax requires either a single box name or a range of box names. The **display** command displays on the screen the current contents from the record buffers of the screen boxes named. The **input** command positions the cursor at the beginning of the first box specified and awaits input.

The **clear** command clears the contents of the specified boxes, if one or more boxes are listed. With no box list, all screen boxes and the message area are cleared and the record buffers are re-initialised, with the exception of the record buffers of any files that have been saved using the **preserve** command. The **newform** command clears all screen boxes and re-displays an empty screen form.

The **highlight** command is similar to the **display** command, but displays the data with special screen highlighting, providing the terminal supports this feature.

An example of an input command to a range of boxes:

```
        input c_code-c_addr4
```

The cursor will wait at each box in turn, from c_code to c_addr4, for input to be made. Pressing the RETURN key moves on to the next box.

PROGRAM FLOW CONTROL COMMANDS

These commands are used to control the sequence of processing statements within the program. The commands include:

```
        goto, gosub, return, if...then...else, scroll
```

The **goto** command transfers control to the specified line label. **gosub** calls a subroutine, which ends when a **return** command passes control back to the main program. The **if...then...else** command sets up a conditional test which can be followed with the desired subsequent statements. The **scroll** command is used to control the input or display of fields within a scroll area on the screen.

For example: **if code = 0 then gosub NEWACC**

MESSAGE DISPLAY COMMANDS

These commands deal with the display of error messages, information messages and prompts. The commands are:

```
        error, message, prompt
```

The **error** command is normally used when an unacceptable condition has been identified in the program. It is usually followed by a **goto** command to pass control to the appropriate part of the program for correction of the error. The **message** command simply displays the specified text expression at the bottom of the screen. The message remains displayed until another **message** command is issued, or a **clear** command without a box list is made. The **prompt** command displays a text expression and waits for a response of **y** or **n**. Either response can be used to pass control to a specified line label. Although not one of the message display commands, the **sleep** command is often used in conjunction with the **message** command to suspend execution for a number of seconds to permit the user to read the message before a **clear** command is issued.

For example:

```
message "Customer record deleted"
sleep 5
clear
```

The message will be displayed for five seconds before the screen is cleared.

END COMMANDS

These commands are: **end, exit**

The **end** command is used to end the processing within an option and pass control back to the option prompt. The **exit** command is used to terminate the program altogether and return to the state from which the **sage** program was called. Every option must terminate with an end command to prevent processing from continuing through into the statements for the next option. Most programs will include an **exit** command, as this is the usual way to leave a **sage** program. The only other normal termination is via a **chain** command. Uncontrolled termination of a program, such as switching off the computer, can damage the data files. **exit** is often included as a simple option:

```
*e=exit
   exit
```

PROGRAM LINKING COMMANDS

These commands interrupt processing to call another program or task. The commands are:

chain, exec, execu

The **chain** command terminates the **sage** program, while the **exec** and **execu** commands return control to the **sage** program when the execution is completed, **execu** being used to call a task which does not require screen display. (**Note:** Because these commands interface with operating system facilities, the precise details of their operation will vary slightly depending on the operating system in use.)

FIELD VALUE COMMANDS

These commands are used to assign a specified value to a field, or to obtain a string with a specified value. The commands are:

let, setstr, getstr

The **let** command designates a field and then assigns a value, either by means of an expression, or by the name of another field, or a combination of the two. The word **let** is optional and is usually omitted. The **setstr** command overwrites a portion of the destination field with a specified string of characters from the source field. The **getstr** command obtains a string of characters from a designated portion of the source field.

An example of a **let** command with the word **let** omitted:

```
o_value = o_qty * st_price
```

This command assigns to the field 'o_value' the result of multiplying the contents of the field o_qty by the contents of the field st_price.

DATE COMMANDS

These commands are used to decode and encode day number values for the special date fields. The commands are:

```
decdate, encdate
```

decdate decodes the day number from the designated date field and stores the appropriate figures in the special temporary fields **day**, **month** and **year**, which must have been declared. **encdate** does the reverse and encodes a day number for the designated field from the current values in the **day**, **month** and **year** fields. The syntax is simply the command followed by the date field name, e.g.

```
decdate duedate
```

PROGRAM ABORT CONTROL COMMANDS

These commands are used to enable or disable the two types of program abort control. The commands are:

```
cancel, interrupts
```

If **cancel** is ON, the operator can abort any input operation by pressing the CANCEL key. If **cancel** is OFF, the CANCEL key is ignored. If **interrupts** are ON, the **sage** program will be aborted on receipt of a standard keyboard interrupt; otherwise keyboard interrupts are ignored. The syntax is simply the command followed by the word ON or OFF, e.g.

```
cancel off
```

SAGE EXPRESSIONS AND OPERATORS

Many of the **sage** commands can make use of operators to form complex expressions using keyfile fields, temporary fields and constants. The available operators are as follows:

ARITHMETIC

+	add
-	subtract
*	multiply
/	divide
%	modulus (remainder after integer division)

STRING (i.e. for alphanumeric fields)

+	concatenation (keep trailing spaces)
/	concatenation (strip trailing spaces)
bw	begins with
ct	contains

RELATIONAL

=	equal to
< >	not equal to
<	less than
>	greater than
< =	less than or equal to
> =	greater than or equal to

LOGICAL

and	logical ''and''
or	logical ''or''

The precedence taken by these operators when used in combination with each other is defined in Section 3.5 of the SCULPTOR Reference Manual. The sequence of evaluation can also be controlled by the use of parentheses in the normal way. The following examples show some uses of operators and expressions with **sage** commands:

1) **if c_bal > c_crlim then**
   message "Credit limit exceeded"

2) **name = forename / " " + surname**

3) **if c_code bw "S" or c_code ct "X" then\
   gosub STANDING_ORDER**

4) **vat = (qty * price + carr) * 0.15**

TRAPPING ERRORS

If the **sage** interpreter detects an error, such as a non-existent record or no record selected, it displays an error message and performs a suitable default action, such as returning control to the option prompt. By using trap clauses, however, you can intercept the error and designate your own line label to which control should be passed in the event of an error.

Trap clauses are included in the syntax of the command during which the error would be detected. For instance, a file access command could include a trap clause in case the specified record does not exist. The general syntax for any trap clause is:

<trap code> = <label>

The available trap codes are listed below:

TRAP CODE	MEANING
bs	BACKSPACE key pressed in first character of box
eoi	End of input key pressed
ni	No input entered for a box
nrs	No record selected for the check, delete or write commands
nsr	No such record found by file access command
re	Record exists for insert or write commands
riu	Record in use (by another user)
yes	Response of ''y'' or ''Y'' to the prompt command
no	Response of ''n'' or ''N'' to the prompt command

For example:

read custmrs nsr=M1 riu=M2

where M1 and M2 are the labels of the lines to which control is to be passed.

The trap codes that can be used with each command are specified in the syntax for the command: refer to Section 3.10 of the SCULPTOR Reference Manual for details of individual commands.

PROGRAMMING TECHNIQUES

Appendix A, at the end of this manual, provides an example of a complete system of SCULPTOR programs, including both screen form and report programs. Not all of the commands can reasonably be demonstrated in a simple example, but a selection of the more common uses are included.

4.5 COMPILING THE sage PROGRAM

When you have completed the writing of the program, you will have a text file with a name terminating in a **.f** extension. This is your source code program. For the **sage** interpreter to read your program, this text file will have to be compiled using the compiler program **cf**. The syntax for calling the compiler is:

```
cf <program name>
```

For example, if your program source code file is named ''orders.f'', the call to compile it would be:

```
cf orders
```

(You do not actually have to type the ''.f'' as **cf** assumes it.) **cf** also requires access to the data dictionaries (the **.d** files) for all keyfiles and alternative record descriptors that have been declared in the **sage** program using the **!file**, **!cfile** and **!record** declarations.

If the compilation is successful, the **cf** program creates a file with the same name as the source code file, but with a **.g** extension. This is the object code program which is used by the **sage** interpreter when you actually run your **sage** program.

If the compilation is not successful, the **.g** file is not created (or, in the case of a re-compilation, the existing **.g** file is left unaltered) and error messages are output to identify the program errors. These error messages identify the line number containing the error and the part of the line that caused the error condition. You can then correct the condition by amending your source code file, and then calling the **cf** program again.

4.6 RUNNING THE sage PROGRAM

After a successful compilation, your **sage** program is now ready to run. The command line to run your **sage** program is as follows:

```
sage <program name>
```

For example, if your program name is "orders", the command line is:

```
sage orders
```

(You do not need to type the **.g** extension as **sage** assumes it.) The other method of calling the **sage** interpreter is to select Option 4 on the development menu. The system then prompts for your program name, as above.

The system will then display the screen form that you designed, with the heading at the top, the data boxes in the central screen area, and the menu line at the bottom of the screen. You can now select an option by typing the relevant code and pressing the RETURN key. If the option requires input from you, the cursor will wait at the start of the box where the input is required. Your input should be typed normally, terminating each input with the RETURN key. The BACKSPACE key can be used in the middle of a box to delete the last character entered. When used at the start of a box in a range of inputs, the BACKSPACE key moves the cursor back to the previous box.

INPUT VALIDATION

To guard against the input of unacceptable data, SCULPTOR applications enable input validation in a number of ways. Firstly, there is the validation by field type which is defined in the data dictionary. Secondly, you can add to this by including a validation list to the field descriptor (refer to Section 2.4 of this manual for details). You can also create your own input validation tests using the available **sage** commands in the program. One such method is to validate by conditional test, using the **if...then** command. For example:

```
input arrdate,depdate
if arrdate > depdate then\
    error "Departure date before arrival date!":\
    goto I1
```

CHAPTER 5 REPORT PROGRAMS (sagerep)

Report programs are used to extract information from the keyed files and produce printed reports. They can also be used to perform batch processing routines, and the reports can be organised to calculate and print sub-totals and end of report figures. Records can be selected for inclusion in a report in a number of different ways, giving the programmer complete flexibility in the selection process.

A standard report program can be created using the **rg** program, as described in Section 3.2. However, such a standard report cannot include any of the more advanced features offered by the **sagerep** program, such as multiple cross-reference files, record selection and exclusion, and the accumulation of total, minimum and maximum values for fields. If you want to take advantage of the advanced features, it is necessary to write your own source code program using the SCULPTOR report language.

The source code program is written using whichever text editor is available on your system. When the program is complete, you use the compiler program **cr** to read your source code text file and create the object code program for use at run time.

This chapter describes the structure of **sagerep** programs and explains the use of the SCULPTOR report language commands. Complete details of writing **sagerep** programs can be found in Chapter 4 of the SCULPTOR Reference Manual.

This chapter is divided into six sections:

5.1 REPORT PROGRAM STRUCTURE

A printed report will consist of particular fields arranged on the printed page, the fields being obtained from selected records from one or more keyfiles. The report will require a title, suitably positioned headings, and accumulated information such as totals at the end of each group of records or at the end of the report. In order to achieve this, the **sagerep** program must perform the following tasks:

- Define a title for the report.
- Identify all the files that will be accessed by the program.
- Specify the selection and exclusion criteria by which records will be obtained from the keyfiles.
- List the page headings and page footnotes to be printed at the top and bottom of each page.
- List the fields that are to be printed and their positions on the page.
- Specify any processing that is to be done, such as reading records from cross-reference files, calculating additional values and printing them.
- Define the final actions to be performed at the end of the report.

Format Definitions

Lines that start with a plus sign (+) are format definitions. These are statements that re-define either the heading or the format of a field, overriding the default value which is taken from the data dictionary.

Declarations

Lines that start with an exclamation mark (!) are declarations. These are statements that define the files to be read, specify temporary constants and variables, define titles and headings, specify the record selection criteria, and perform other initialisation tasks.

Comments

Comment lines can be included anywhere in the program by starting the line with a period: these lines are ignored by the program compiler.

Program Statements

Lines starting with any other character are program statements. If the first word on the line is not recognised as one of the SCULPTOR report language words or a field name, it is taken to be a line label. Multiple statements can be placed on one line by separating them with colons. To continue a statement onto a second line, the first line should be terminated with a backslash (\).

Before considering the way in which these different types of program line are combined to form a **sagerep** program, it is necessary to understand the method which **sagerep** uses to process the data files and produce the report.

SAGEREP PROCESSING METHODS

One keyfile is designated as the main file, or driving file. The processing is based around the automatic reading of the records in this file in index sequential order. Various selection methods can be used to start processing at a particular record, to exclude certain records, or to stop processing after a specified record, but, if no such selection is specified, all the records of the driving file will be read in index sequential order.

For each record, the main statements in the **sagerep** program define the processing that is to take place. This may just be the printing of a selection of fields from the record, but the processing can also include the reading of a cross reference file, the performing of calculations and the updating of fields in the keyed file.

At its simplest, therefore, a **sagerep** program can consist only of a declaration of the file to be read, and a list of the fields to be printed. Most applications, however, will require additional processing such as titles, page headings and footings, and some sort of control and selection of the records to be processed. All these, and other, features can be incorporated into a **sagerep** program by grouping together sets of program statements. Each set of statements is executed at the appropriate time in the program: some at the start of processing, some for every record, some when a particular field value is read, and some at the beginning and end of pages and at the end of the report.

The groups of program statements are as follows:

INITIALISATION STATEMENTS
TITLE STATEMENTS
CONTROL STATEMENTS
PAGE HEADING STATEMENTS
PAGE FOOTNOTE STATEMENTS
''ON STARTING'' STATEMENTS
''ON ENDING'' STATEMENTS
FINAL STATEMENTS
MAIN STATEMENTS

The diagram on page 5-5 shows the position in the **sagerep** processing cycle of these groups of statements.

The initialisation and title statements are executed first, before the driving file is read. Then a record is read from the driving file and the control statements are executed. If the control statements indicate that the record should be excluded, processing of the record terminates and the next driving file record is read.

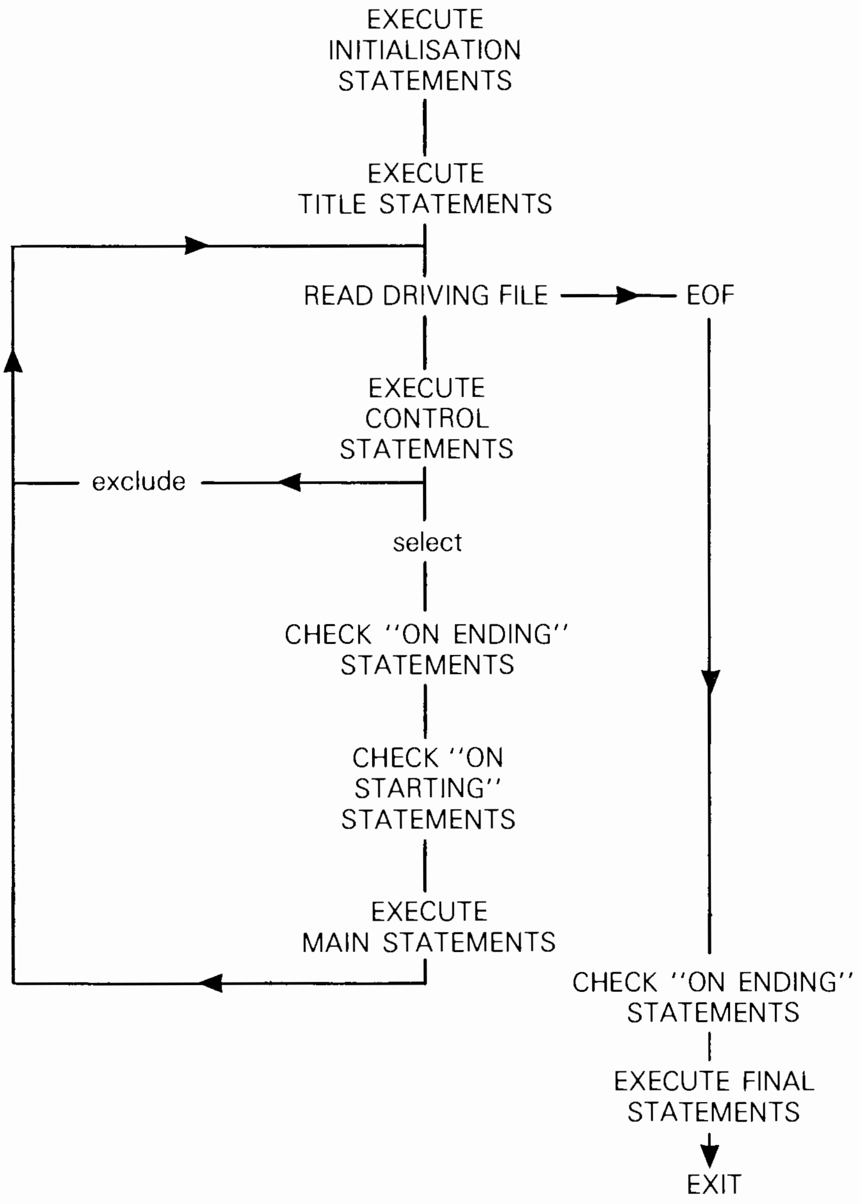
The ''on ending'' and ''on starting'' statements are checked. If the record meets the condition for the start or end of a group, the necessary processing is performed.

Next, the main statements are executed for the record. Page heading and footnote statements are executed as required whenever the top and bottom of a page is encountered. The cycle then returns to read another record from the driving file.

When the end of file is reached on the driving file, the ''on ending'' statements are checked, and last of all the final statements are executed.

SAGEREP PROCESSING CYCLE

sagerep Program


EXECUTE INITIALISATION STATEMENTS
EXECUTE TITLE STATEMENTS
READ DRIVING FILE
EOF
EXECUTE CONTROL STATEMENTS
exclude
select
CHECK "ON ENDING" STATEMENTS
CHECK "ON STARTING" STATEMENTS
EXECUTE MAIN STATEMENTS
CHECK "ON ENDING" STATEMENTS
EXECUTE FINAL STATEMENTS
EXIT

Because **sagerep** performs the cycle of statement execution automatically, there is no need for the main statements to include a command to read from the driving file: the cycle is completed for each record in turn without the commands having to create a program loop. The main statements, therefore, need only list the processing and printing commands that are to be performed for each selected record.

The other groups of statements are identified by the declaration used, e.g. **!title, !heading, !on starting**, etc. Within any group, the statements are executed in the order in which they are declared in the program. If required, a declaration can call a subroutine instead of specifying directly the processing or printing to be performed. Subroutines, when used, are identified by line label. If subroutines are used, the main statements must terminate with an **end** command.

A typical **sagerep** program could have a structure as shown below:

+format definitions	Format definitions are only required for fields when a heading or format is required other than that defined in the data dictionary.
!declarations	Declarations are written for all the groups of statements other than the main statements.
main statements	The main statements define the processing and printing that is to be performed for each record read from the driving file.
subroutines	Subroutines, if used, can appear anywhere in the program, but are often grouped together at the end. They are identified by line labels.

The following sections of this chapter explain the format definitions, declarations, and the **sagerep** commands.

5.2 sagerep FORMAT DEFINITIONS

Each field has its description defined in the data dictionary created using the **describe** program. The only parts of the field description that can be altered for printing are the field heading and the format. If no format definition is made for a field, **sagerep** will use the default value from the data dictionary. Format definitions need only be included in a **sagerep** program when you want to alter either the field heading or the format.

Format definitions should normally be placed at the beginning of the program, before any of the declarations. The syntax for a format definition is as follows:

```
+ <field name>,[<heading>][,<format>]
```

Either the heading or the format or both can be redefined. If the heading is to remain as the default but the format is to change, two commas must separate the field name and the format.

For example:

```
+c_name,Name,t
+c_addr1,,t
+c_tel,Telephone No
```

The first field, c_name, has both a different field heading (Name) and a different format specified. (The format code ''t'' causes trailing spaces to be removed from alphanumeric fields. Refer to Section 2.6 of the SCULPTOR Reference Manual for other format codes.) The second field, c_addr1, uses the default field heading (two consecutive commas), but has a new format. The third field, c_tel, has a new field heading, but the format is unchanged.

5.3 sagerep DECLARATIONS

After any format definitions have been specified, the next section of a **sagerep** program is the declarations. Each of the specialised groups of statements consists of one or more declarations, the declaration lines starting with an exclamation mark. The declarations that are available in **sagerep** are listed below. The full explanations of the declarations can be found in Section 4.5 of the SCULPTOR Reference Manual.

INITIALISATION STATEMENT DECLARATIONS

!file Declare the driving file. Only one driving file can be declared. The declaration must specify either the filename or a full pathname. The number 1 can be used to identify the file, this number being used subsequently by any commands that access the file explicitly. For example:

```
1) !file orders
2) !file 1 orders
3) !file /usr/accs/taxfile
```

!init Declare an initialisation statement that will be executed once only at the beginning of the report. The declaration can specify any **sagerep** command except **goto**. For example, the **!init** declaration could be used to call a subroutine to perform a stationery alignment check, e.g.

```
!init gosub ALIGN
```

!temp Declare a temporary field for use within the program. Fields that you declare here can be used in the program and printed in the same way as keyfile fields. The syntax for declaring a temporary field is the same as in the **describe** program, with the exception that no validation list is allowed. Additionally, the declaration can end with an assignment similar to that used in the **let** command. Refer to Section 2.4 of this manual for further details of the **describe** program. For example:

```
!temp total,TOTAL, m4
!temp value,Value,m4 = o_qty * st_price
```

!display	Display a message on the screen, for example: `!display "OUTSTANDING ORDERS REPORT"`
!input	Input an initial value into a temporary field. The specified prompt text is displayed on the screen (followed by a question mark), and the subsequent input is stored in the designated field. For example: `!input "Stock Code",tmp_stock`
!constant	Similar to **!temp**, this declaration specifies a temporary field and its initial value. For example: `!constant status,,i1 = 1`

!read	Declare a cross-reference file and initially read a record from it. The key value is taken from the field list specified in the **key=** clause. The values in these fields must have been established in a previously declared initialisation statement. For example:
	`!constant ckey,,a1 = "A"`
	`!read control key=ckey`

!startrec	Define the record in the driving file at which the report is to start. The key value is taken from the field list specified in the **key=** clause. The values in these fields must have been established in a previous declared initialisation statement. For example:
	`!temp tmp_stock,,a6`
	`!input "Stock Code",tmp_stock`
	`!startrec key=tmp_stock`

!endrec	Define the record in the driving file at which the report is to end. The key is defined in the same way as for **!startrec**.

TITLE STATEMENT DECLARATIONS

!title Declare a title statement that will be executed once only, after the initialisation statements, but before the start of the report. An obvious example is to print the title of the report and one or two blank lines, but **!title** statements can also be included to perform setup routines, print the date, print an introductory paragraph, and so on. There is no limit to the number of **!title** statements, and the only **sagerep** command that cannot be used in a !title statement is **goto**. For example:

```
!title print "OUTSTANDING ORDERS REPORT": print
```

(Prints the title and a blank line.)

CONTROL STATEMENT DECLARATIONS

!cfile Declares a cross-reference file which is initially closed. The declaration must specify either the filename or a full pathname. The file can be given a file number, starting at 2, which can be used by commands in the program that access the file. For example:

```
!cfile 2 charges
```

!xfile Declares a cross-reference file which is initially open. The declaration must specify the file as for **!cfile**, above. If a **key=** clause is included, a control statement is created to automatically read a record from the cross-reference file each time a record is read from the driving file. The maximum number of cross-reference files (declared either by **!cfile** or **!xfile**) is 15. For example:

```
!xfile 3 custmrs
!xfile 4 stock key=o_stkno
```

!record Declare an alternative record layout for a file. The file must already have been declared, and the alternative layout must have been defined using the **describe** program. e.g.

```
!record custmrs custmrs1
```

!select and **!exclude**

These two declarations are used to set up the conditions for including records from the driving file in the report. When each record is read, it is tested against the specified conditional expressions and then either selected or excluded depending on the result. The priority of the conditions is determined by the order in which you place the **!select** and **!exclude** declarations. For example:

```
!select if duedate <= date
!exclude if c_code = "S" or c_code = "X"
!select if ordcode > 0
```

This set of conditions will select records where the duedate field has a value less than or equal to today's date, regardless of the other two conditions. Records that do not meet this first condition will be excluded if the c_code field is S or X. The remaining records will be included if the ordcode field is greater than zero. All other records will be excluded.

Any number of **!select** and **!exclude** declarations can be made in a program. Refer to Section 4.5 of the SCULPTOR Reference Manual for further details of the way in which **sagerep** interprets the selection and exclusion conditions.

NOTE: Other techniques for selecting records for processing include the following:

- Using **!startrec** and **!endrec** declarations.
- Inputting parameters from the terminal at run time.
- Reading parameters from a file read at run time.
- Passing arguments to the program from the command line entered at run time.

These techniques are explained in detail in Chapter 4 of the SCULPTOR Reference Manual.

HEADING STATEMENT DECLARATIONS

!heading Heading statements are executed, in the order in which they are written in the program, at the start of each new page. Any **sagerep** command can be specified except **goto**. The most common use is to print column headings at the top of the page, using the **printh** command to align the headings with the field data to be printed for each record. For example:

```
!heading printh *c_name,*c_ordno,*o_date,*o_value:\
         print
```

FOOTNOTE STATEMENT DECLARATIONS

!footnote Footnote statements are executed, in the order in which they are written in the program, each time the report reaches the foot of a page. Any **sagerep** command can be specified except **goto**. For example:

```
!footnote print tab(40); "Page: ";pageno
```

ON STARTING STATEMENT DECLARATIONS

!on starting Declares a statement to be executed when a new value is detected in the specified field. Any number of ''on starting'' declarations can be made, execution being in the order in which they appear in the program. Any **sagerep** command can be specified except **goto**. For example:

```
!on starting s_areacode newpage
```

ON ENDING STATEMENT DECLARATIONS

!on ending Declares a statement to be executed on ending a value in the specified field. Any number of ''on ending'' declarations can be made, execution being in the order in which they appear in the program. Any **sagerep** command can be specified except **goto**. For example:

```
!on ending c_ordno gosub END_ORDER
```

FINAL STATEMENT DECLARATIONS

!final Declares a statement that will be executed once only at the end of the report. Any number of final statements can be made, execution being in the order in which they appear in the program. Any **sagerep** command can be specified except **goto**. For example:

```
!final print: print "END OF REPORT"
```

PRINTOUT CONTROL DECLARATIONS

In addition to the declarations for the various types of program statement, there are three declarations that can be used to alter the format of the printout of the report. These are **!depth, !width** and **!gap**. The first two define the number of lines and columns to be used for the report. The !gap declaration is used to alter the standard space between printed items.

For a simple application, many of these declarations will not be required. A straightforward report program reading from a driving file and two cross-reference files, with a temporary field for totals, printing the title, page headings and footnotes, and final totals, could be produced using only the following declarations:

```
!file 1 filename_a
!temp total,Total,m4

!xfile 2 filename_b key= a_code
!xfile 3 filename_c key= b_custno

!title print tab(30);"REPORT TITLE": print
!heading gosub HEADING
!footnote gosub PAGENO
!final gosub END_REPORT
 .
 .
```

followed by the Main Statements and the commands for the subroutines.

5.4 sagerep COMMANDS

When the format definitions and declarations have been completed, the remaining part of a **sagerep** program consists of the main statements and any subroutines. This is the part where you define the processing and printing that is to take place in the program. The main statements and subroutines, and many of the declarations, make use of the commands that are available in the SCULPTOR report language.

Most of the **sagerep** commands are very similar in both syntax and method of use to their counterparts in the **sage** language. To avoid repetition, these commands are not further explained in this section. You can refer to Section 4.4 of this manual for a brief overview of the **sage** commands, and a complete list of the available **sagerep** commands is given at the beginning of Section 4.6 of the SCULPTOR Reference Manual.

The following paragraphs briefly explain the commands that are unique to **sagerep** and are not used in **sage**.

PRINT COMMANDS

The two commands that are used to print items in the report are:

print and printh

The **print** command prints the listed items, which can include the contents of a named field, the heading of of a field, a constant, a blank line, a space the same size as the named field, and several other items designed to make report printing both easy to control and extremely flexible. The **printh** command operates in much the same way, but it also adjusts the spacing to ensure that the data fields are aligned with the column headings. (To achieve this, it is necessary to make sure that the **printh** command for the headings includes the field headings for exactly the same list of items as the **printh** command list for the data field values.)

Both the print commands can also specify characters to start and stop the printing of special effects, such as double width characters, underlining, or tabbing to a designated column. By specifying **total**, **min** or **max** and a field name, you can print the current accumulated total, the minimum or the maximum value for that field. The complete list of items that can be included in a **print** or **printh** list is given under the **print** command in Section 4.6 of the SCULPTOR Reference Manual.

For example:

```
!heading printh *st_code,*st_descr,*st_unit,*st_qty,*st_price:\
        print

printh st_code,st_descr,st_unit,st_qty,st_price
```

The **!heading** declaration will print the field headings (specified by the preceding asterisks) for the list of fields, followed by a blank line. The use of the **printh** command ensures that the subsequent printing of the data content of the fields will be aligned with the field headings.

PRINT CONTROL COMMANDS

These commands provide additional control of the printout. The commands are:

```
keep, newpage, width
```

The **keep** command checks the number of lines left on the current page. If the number left is less than the specified value, a new page is started. The **newpage** command is used to force the start of a new page at a particular point in the program. The **width** command changes the current line width to the specified number of columns.

For example, if the main statements will produce six lines of printed output for each record processed and it is required not to split a record over two pages, the main statements can be started with the command:

```
keep 6
```

ABORT COMMAND

In addition to the **exit** command which terminates the program after executing the **!final** statements, **sagerep** includes the **abort** command which terminates the program immediately without executing the **!final** statements. Control is passed to the task from which the **sagerep** program was called.

SAGEREP EXPRESSIONS AND OPERATORS

Many of the **sagerep** commands can make use of operators to form complex expressions. The available operators are the same as those used in **sage** programs and their use is explained in Section 4.4 of this manual.

Appendix A, at the end of this manual, provides an example of a complete system of SCULPTOR programs, including both screen form and report programs. Not all of the declarations and commands can be demonstrated in a simple example, but a selection of the more common uses are included.

5.5 COMPILING THE sagerep PROGRAM

When you have completed the writing of the program, you will have a text file with a name terminating in a **.r** extension. This is your source code program. For the **sagerep** interpreter to read your program, this text file will have to be compiled using the compiler program **cr**. The syntax for calling the compiler is:

```
cr <program name>
```

For example, if your program source code file is named ''custmrs.r'', the call to compile it would be:

```
cr custmrs
```

(You do not actually have to type the ''.r'' as **cr** assumes it.) **cr** also requires access to the data dictionaries (the **.d** files) for all keyfiles and alternative record descriptors that have been declared in the **sagerep** program using the **!file, !cfile, !xfile, !read** and **!record** declarations.

If the compilation is successful, the **cr** program creates a file with the same name as the source code file, but with a ''.q'' extension. This is the object code program which is used by the **sagerep** interpreter when you actually run your **sagerep** program.

If the compilation is not successful, the **.q** file is not created (or, in the case of a re-compilation, the existing **.q** file is left unaltered) and error messages are output to identify the program errors. These error messages identify the line number containing the error and the part of the line that caused the error condition. You can then correct the condition by amending your source code file, and calling the **cr** program again.

5.6 RUNNING THE sagerep PROGRAM

After a successful compilation, your **sagerep** program is now ready to run. The command line to run your **sagerep** program is as follows:

```
sagerep <program name> [<ppf> <arguments>]
```

where **<ppf>** is a printer parameter file, and **<arguments>** are parameters to be referenced by the special temporary field **arg**. These last two elements of the **sagerep** command line are optional and are described in full detail in Section 4.7 of the SCULPTOR Reference Manual.

By default, the output will be directed to the terminal. If this is the desired output destination, the printer parameter file **pvdu** should be specified. To print the report, the output should be redirected to the required printer, and the appropriate printer parameter file should be specified.

NOTE: The printer parameter file does not cause the output to be sent to any particular device. Redirection of the output must be achieved using the operating system's I/O redirection facilities.

For example, to print the report of the **sagerep** program "custmrs", using the printer parameter file **p80** under the Unix operating system, the command line would be:

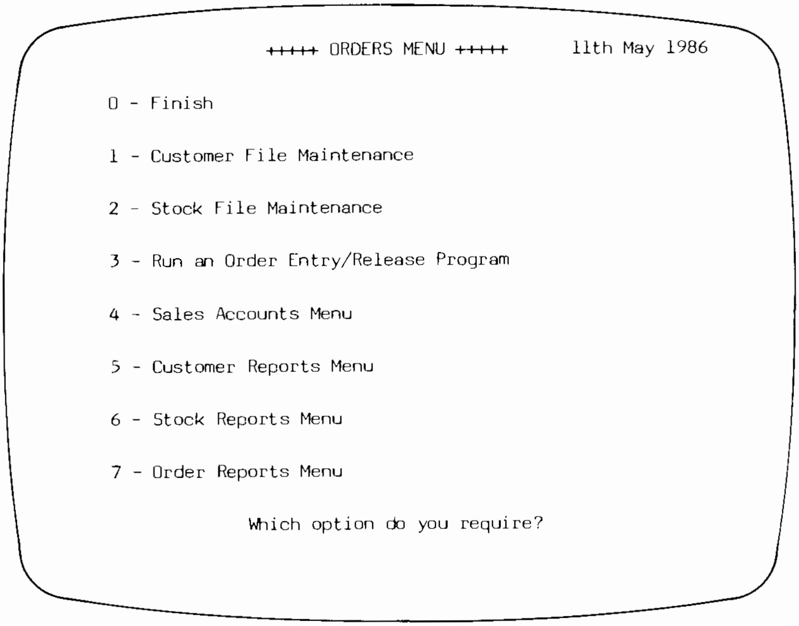
```
sagerep custmrs p80 >/dev/lp
```

(You do not need to type the ".q" extension as **sagerep** assumes it.) Refer to your operating system instructions for details of the I/O facilities.

The other method of calling the **sagerep** interpreter is to select either Option 6 or Option 7 on the SCULPTOR Development Menu, depending on whether you want the output directed to the terminal or the printer. The system then prompts for your program name, as above.

CHAPTER 6 THE MENU PROGRAM

The **menu** program is an easy to use part of the SCULPTOR system that enables you to produce neatly formatted menus on the screen that can be used to access your SCULPTOR application programs. A typical menu screen produced by the **menu** program is shown below:


```
                 +++++ ORDERS MENU +++++      11th May 1986

     0 - Finish

     1 - Customer File Maintenance

     2 - Stock File Maintenance

     3 - Run an Order Entry/Release Program

     4 - Sales Accounts Menu

     5 - Customer Reports Menu

     6 - Stock Reports Menu

     7 - Order Reports Menu

                 Which option do you require?
```

All SCULPTOR produced menus will have this type of format with a title line and the system date at the top of the screen, followed by a numbered list of options, and the prompt asking you to select an option at the bottom of the screen. The options can be **sage** programs, **sagerep** programs, other menus, or any other programs or directories that can be accessed in your system.

The **menu** program consists simply of a text file interpreter. The code for your menu is written as an ordinary text file using whatever text editor is available on your system. The text file must be given a ''.m'' extension and the code should be written in the following format:

FIRST LINE

The menu title to appear at the top of the screen.

SUBSEQUENT GROUPS OF LINES

Each option requires two lines of code, the first contains the option number and the text to be displayed; the second contains the program command (or commands) to be executed when the option is selected. A further enhancement can be included by using the parameter substitution character **%** in the command line. This enables the user to input the actual value of the parameter (for example, the program name) when the option is selected. If this feature is used, a third line is required to specify the prompt to be displayed when the option is selected.

The **menu** program supplies the system date and the prompt for the bottom of the screen. An option ''0 - Finish'' is provided automatically by the menu program, but an alternative option for 0 can be created if required. There must always be an option with the command line **exit** in order to provide a way out of the menu.

An example of the parameter substitution feature can be seen in Option 3 of the menu shown on page 6-1. When this option is selected, the system displays the prompt:

Which program do you want to run?

When the program name is entered, the command line for option 3 is then executed.

Page 6-3 shows the text file for the sample menu. Further details of the **menu** program can be found in Section 5.4 of the SCULPTOR Reference Manual.

The text file (orders.m) for the Orders Menu:

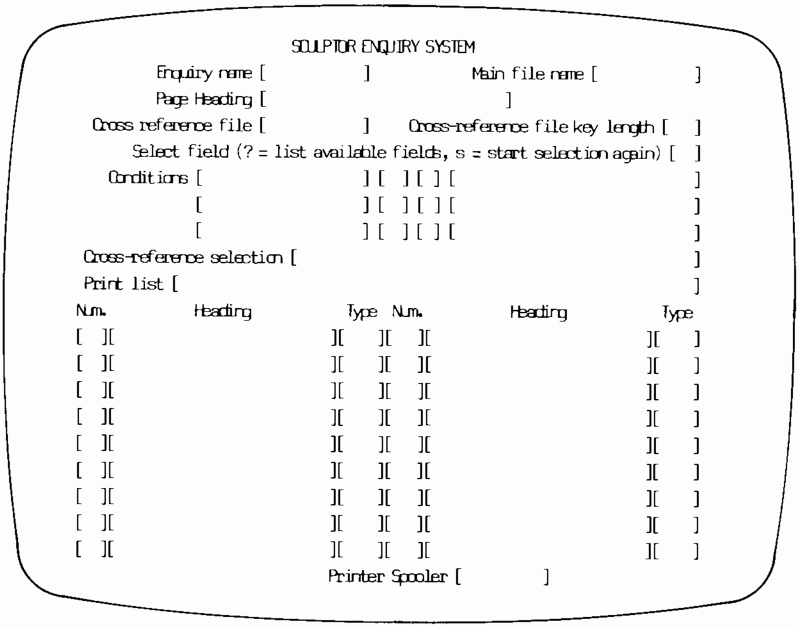
```
ORDERS MENU

1,Customer File Maintenance
sage custmrs
2,Stock File Maintenance
sage stock
3,Run an Order Entry/Release Program
sage %
Which program do you want to run
4,Sales Accounts Menu
menu sales
5,Customer Reports Menu
menu custreps
6,Stock Reports Menu
menu stkreps
7,Order Reports Menu
menu ordreps
```

CHAPTER 7 THE QUERY PROGRAM

The **query** program provides a very straightforward method of making specific enquiries into SCULPTOR keyed files and producing ad hoc reports of the results of the enquiries. Access to the **query** program is achieved by typing:

sage query

The system then displays a blank enquiry screen, as shown below:



```
                    SCULPTOR ENQUIRY SYSTEM
        Enquiry name [           ]          Main file name [          ]
        Page Heading [                           ]
  Cross reference file [         ]   Cross-reference file key length [  ]
      Select field (? = list available fields, s = start selection again) [  ]
    Conditions [                 ][  ][ ][                          ]
               [                 ][  ][ ][                          ]
               [                 ][  ][ ][                          ]
 Cross-reference selection [                                         ]
 Print list [                                                        ]
Num.        Heading           Type  Num.         Heading          Type
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
[  ][                        ][    ][  ][                        ][    ]
                      Printer Spooler [         ]
```

By responding to a series of prompts, you complete the empty boxes on the screen and so give the **query** program the required information to enable it to interrogate the specified keyed files and extract the data for the printed report.

The first prompt that you have to respond to is the enquiry name. The first time you make a particular enquiry, this will be a new name and you will have to respond to the remaining prompts to provide the names of the files, the selection criteria for records to be included in the query report, the list of fields to be printed, and so on. On subsequent occasions, entering the name of an existing enquiry will cause the **query** program to use the previously defined responses.

The enquiry can be into a single file (defined in the ''Main file name'' box), or can also include a single cross-reference file, in which case you then have to specify the fields for cross-reference file matching.

In addition to the prompts for filenames, record selection conditions and the print list, the **query** program also asks if you want to include a totals line in the report, if you want to run the enquiry immediately (or merely store the selections), and if you want to direct the output to the screen or the printer.

Refer to Section 5.10 of the SCULPTOR Reference Manual for further details of the prompts and responses in the **query** program.

CHAPTER 8 THE UTILITY PROGRAMS

To support the main SCULPTOR programs which have been described in the earlier chapters of this manual, the SCULPTOR system also includes the following utility programs:

- The keyed file utilities — **kfcheck, kfcopy, kfdet** and **kfri**
- **reformat**
- The Language Configuration program — **lcf**
- The VDU parameter file programs — **setvdu/decvdu**
- The printer parameter file programs — **setprinter/decprinter**
- **sageform**

THE KEYED FILE UTILITIES

The keyed file utilities are used to maintain SCULPTOR keyed files.

kfcheck	The Keyed File Integrity Check program is used to perform a complete check on the index of a data file.
kfcopy	The Keyed File Copy program is used to copy an existing keyed file to a new filename.
kfdet	The Keyed File Details program is used to display the key length, record length and number of index levels for a selected keyed file.
kfri	The Rebuild Keyed File Index program is used to build a new index from an intact data file that has had its index damaged.

Refer to Section 5.1 of the SCULPTOR Reference Manual for details of running these programs.

REFORMAT

The **reformat** program is used to reorganise a keyed file when the record structure has been changed. If you change the record length of a file, alter the size or type of any of the fields, or insert or delete fields within the record, it is essential that the file is reformatted in order to avoid corrupting the existing data in the file. You will also have to recompile all the **sage** and **sagerep** programs that access the file. Refer to Section 5.2 of the SCULPTOR Reference Manual for details of the **reformat** program.

THE LANGUAGE CONFIGURATION PROGRAM

The program **lcf** is used to provide alternative versions (for example, in another language) of the standard text items and special characters that are displayed in the SCULPTOR programs. For example, the standard text displayed on **sage** screen forms can be translated, the default box delimiters can be changed, or the format for displaying the date can be altered. Refer to Section 5.3 of the SCULPTOR Reference Manual for details of the **lcf** program.

VDU PARAMETER FILES

In order to be able to accept keyboard input and display items on the screen, SCULPTOR must be supplied with the VDU parameters for your terminal. A set of VDU parameter files is supplied with the SCULPTOR system, and, for most systems, one of these will define the parameters for your system. However, in rare cases that are not covered by one of the supplied files, you can create a new VDU parameter file using the program **setvdu**. The associated program **decvdu** can be used to print the contents of an existing VDU parameter file. Section 5.8 of the SCULPTOR Reference Manual contains full details of these programs.

PRINTER PARAMETER FILES

Similarly to the VDU parameters, SCULPTOR must also be supplied with a printer parameter file. Again, a set of printer parameter files is supplied with the SCULPTOR system, and one of these may match your printer. If not, a new printer parameter file can be created using the program **setprinter**. The associated program **decprinter** can be used to print the contents of an existing printer parameter file. Refer to Section 5.9 of the SCULPTOR Reference Manual for further details.

SAGEFORM

This program is used to produce a printed version of a **sage** screen form, for use in your program documentation, for example. The specified program must be a compiled **sage** program, and **sageform** gives you the opportunity to indicate the screen size in rows and columns (defaulting to 80 by 24). The output can either be printed, or just displayed on the screen.