CHAPTER 4

IMPLEMENTATION OF STATISTICAL EXPERT SYSTEM FOR INFORMATION MANAGEMENT (SESYS)

4.1 Introduction

The second and third chapters have accomplished the task of designing and developing a Statistical Expert System, in all its requisite features, which could be both advice-giving and pattern-finding. In other words, it advises both the untrained and trained users on the methods of statistics that they could use to analyse the kind of data they have collected for specific purposes. It also directs them to seek the help from the existing statistical packages such as SPSS, STATISTICA, and SYSTAT for analysis or use the SESYS to process data and interpret. It is not beyond the SESYS to recognise patterns in the data being used in the analysis and interpretation. The conceptualisation and the conceptual clarifications have already been accomplished, in the last two chapters, leaving the implementation of the SESYS for this chapter.

The present chapter therefore discusses in detail the implementation of the SESYS modules, input-output programs facilitating consultation and developing and using dynamic databases based on the organisational types and requirements of the users and the data types. This permits also the use of the resources in the most efficient/elegant way, avoiding any wastage and wasteful expenditure of time and energy. The chapter then focuses on interfacing the SESYS with ‘C’ routines, by applying direct call technique.

Some specific routines, which could, for example, compute index numbers such as the LASPEYRES index, towards information management, have been discussed in order to inform about the capabilities of the SESYS.
The SESYS program is linked with the 'C' routines and for invoking the statistical packages, according to the requirements of the users. Lastly, the chapter presents two dry runs of samples, namely, the KBSSP and the SESA. By the dry runs, the SESYS is in a way evaluated for the purpose of letting the potential user understand and take advantage of the capabilities of the SESYS.

**Implementation of SESYS Modules.** This section describes the modifications we may make to independent modules created during SESYS program construction so that those modules work well together in a SESYS project.

First, we look at the general considerations regarding problems that often arise when converting from standalone modules to an integrated project. Then we examine the main driver routine in the SESYS Project. Next, we determine what if anything must be changed in each of the SESYS modules. All of this work prepares us for compiling and linking the SESYS project. Finally, we compile, link and execute the SESYS project.

After we have implemented all modules in a Turbo Prolog project and also when the time comes to pull the independently tested pieces together, we will probably need to make changes to the modules. In real world applications, we often need to include some code segment in a module for testing. But when the project is assembled, not only is the code unnecessary, it may also cause an error (Shafer, 1987).

Another type of change we see as we progress from independent module towards a fully integrated and operational project is in main the driver routine. For example, we might define the driver routine to include a predicate that requires a list of arguments. Later, as we debug the routine, we find that we want to pass the list of arguments not at the time the routine is invoked but rather later. We would have to change the main driver routine so it does not create an error during compilation or at run time.
The primary problems when we prepare modules for project compilation and linkage are conflicts and duplication. It is important to remember that a finished Turbo Prolog project is in some ways a single program and in other ways a collection of independent programs. Learning when and how to use the project in the appropriate way is something that comes only with experience.

The SESYS requires one and only goal statement. The goal statement is the first predicate executed when the module is run.

The method known unknown predicate appears in the STATISTICAL KNOWLEDGE BASE (SKB) module, the get response predicate is in the QUERY module, which is Check method predicate is in the REASON module and the time to stop predicate is in the QUIT module. This is the order in which the modules are normally executed.

Because the main driver module "knows" about the global predicates but does not explicitly load individual modules, we cannot test it as a standalone module. In effect, it is not a standalone module. If we try to load and run it like any other module, we will encounter an error 401: no clauses for this predicate. This is because the first predicate in the definition of our goal clauses start is method known unknown and the main driver module does not have access to the Statistical Knowledge Base (SKB) module in which that predicate clauses appear.

We keep the driver routine small. So that testing it independently is unnecessary.

Modifying the SESYS Modules. We have included the names of all six modules in the SESYS.PRJ file; no further modification of that file is needed. Until compilation and linkage take place, this file is not used. All the duplicate and individual declarations are removed and they are replaced with one global predicate declaration. When we declare any predicate that
includes arguments in the GLOBDEF.PRO file, we provide one or more flow patterns for the parameters, whenever necessary.

In GLOBDEF.PRO, we do not need to make any changes because we have been building the global declarations as we progressed through the design. We include the dynamic statistical knowledge base declaration within the GLOBDEF.PRO file. Delete the knowledge base declaration section of the statistical knowledge base module. No other changes are required.

**Stopping the SESYS Program.** If we compile SESYS now, it performs all the tasks defined in the program, and exits the program. If we want to get SESYS recommendations more than once, the program has to be run repeatedly. This is because we have not provided a “Trigger”, or means to notify the program when to stop. As a result, the SESYS quite logically executes each of the clauses in the driver module in turn.

Because the `check_method( technique)` predicate only goes through the process of identifying a statistical method at one time, we must find a way to force that predicate to continue to execute until the user signals that he or she is ready to stop.

There are a number of ways to do this. One is to put in a new predicate that provides only the trigger action. Here, we only need to add a single line to `check_method` predicate, which checks for a specific input from the user.

If this input is encountered, the predicate fails and the `time_to_stop` predicate executes. We can use whatever trigger we like to stop the program. We'll use exit. Here's how this `check_method(Technique)` line will look:

```
check_method( decide) :-
    decide <> "exit",
    method_known_unknown.
```

**SESYS Input and Output.** A non-technical user would have difficulty using the SESYS. We would add more rules and facts, but a larger problem exists, the user interface needs to be changed so that
The user can easily define the statistical goal.

Specific reply is entered by the user in response to prompt at the time of program execution; this is not written into the program.

Results are communicated to the user in English-like sentences. In the following section, we show how to add this kind of friendlier user interface to the SESYS program.

4.2 The SESYS Consultation Paradigm

Often, we want a Turbo Prolog program to enter into a dialog with the user-to-ask questions, issue responses, and perhaps tell the user how it reached a specific conclusion.

Our aim might be to allow the following type of dialog:

```
What is the Name of the user?
xxxxxxx
Employee Number?
25480
Do you know the Statistical technique?
No
Do you want to use the SESYS program?
Yes
Do you ... ...
```

In the SESYS design, this is called a consultation paradigm. The general structure works with almost any type of diagnostic system. Notice two elements of this structure:

Most of the questions require Yes or No answer, although some are assigned values, such as the user's Name and Employee Number, to input variables (bind the variables) that will be used later in the program. We can also create multi-choice questions.

To add input and output features to the SESYS, we use the built-in predicates that support these features (Refer to Turbo Prolog User's Reference Manual).

Using WINDOWS. Turbo Prolog permits us to create and use windows while executing. This allows us to develop excellent user interfaces
with minimal programming. To clear the dialog window, use the clearwindow predicate. The modified version of the go_ahead clause is

```prolog
go_ahead :-
    clearwindow,
    write("what is the user name ?"),
    readln(username),
    statistical_technique(method),
    write(" the statistical technique is ", method), nl.
```

If we had made all of the changes to the SESYS program discussed in this section, our program should now look like Figure 4.1.

```prolog
/* Begin of Program */
domains
technique, indication = symbol
stat_problem = string
user_id = integer
predicates
statistical_technique(stat_problem, technique)
known(method, indication)
user_response(char)
go_ahead
clauses
go_ahead :-
    write("What is the name of the statistical problem?")},
    write("Otherwise type STATISTICAL PROBLEM"),
    readln(stat_problem),
    write("What is ID-NUMBER?")},
    readln(User_ID),
    statistical_technique(Stat_problem, Technique),
    write(Stat_problem, "needs the ", Technique, "to solve our problem"),
    nl,nl.

go_ahead :-
    write("Sorry, I am unable to find out the statistical technique"),
    write(" required to solve your problem.")},
    write(" Better consult a statistician"), nl.

known(Stat_problem, current_year_prices) :-
    write(" Do you know the current year prices (y/n) ?")},
    response(Reply),
    Reply='y'.

known(Stat_problem, base_year_prices) :-
    write(" Do you know the base year prices (y/n) ?")},
    response(Reply),
    Reply='y'.

known(Stat_problem, current_year_quantity) :-
    write(" Do you know the current year quantity (y/n) ?")},
    response(Reply),
    Reply='y'.
```
known(Stat_problem, base_year_quantity) :-
    write("Do you know the base year quantity (y/n) ?"),
    response(Reply),
    Reply='y'.

known(Stat_problem, quantities_from_a_representative_year) :-
    write("Do you know the quantities from a representative year (y/n) ?"),
    response(Reply),
    Reply='y'.

known(Stat_problem, no_of_elements_used_in_index) :-
    write("Do you know the number of elements used in Index calculation (y/n) ?"),
    response(Reply),
    Reply='y'.

known(Stat_problem, quantities_prices_that_determine_values_we_use_for_weights) :-
    write("Do you know the quantities and prices that determine values we use for weights (y/n) ?"),
    response(Reply),
    Reply='y'.

statistical_technique(Stat_problem, unsigned_aggregatesindex) :-
    known(current_year_quantities),
    known(base_year_quantity).

statistical_technique(Stat_problem, weighted_aggregatesindex) :-
    known(current_year_prices),
    known(current_year_quantity),
    known(base_year_prices).

statistical_technique(Stat_problem, laspeyres_index) :-
    known(current_year_prices),
    known(base_year_prices),
    known(base_year_quantity).

statistical_technique(Stat_problem, paasche_index) :-
    known(current_year_prices),
    known(base_year_prices),
    known(current_year_quantity).

statistical_technique(Stat_problem,fixed_weight_aggregates_pricesindex) :-
    known(current_year_prices),
    known(base_year_prices),
    known(quantities_from_a_representative_year).

statistical_technique(Stat_problem, unweighted_average_of_relatives_priceindex):-
    known(current_year_prices),
    known(base_year_prices),
    known(no_of_elements_used_in_index).

statistical_technique(Stat_problem, weighted_average_of_relatives_priceindex):-
    known(current_year_prices),
    known(base_year_prices),
known(quantities_prices_that_determine_values_we
use_for_weights).

statistical_technique(Stat_problem,
weighted_average_of_relatives_quantityindex):-
known(base_year_quantity),
known(current_year_quantity),
known(quantities_prices_that_determine_values_we
use_for_weights).

response(Reply) :-
readchar(Reply),
write(Reply), nl.
/* End of Program */

Figure 4.1: The SESYS Input and Output Program

4.3 Using Dynamic Database in the SESYS

It is necessary that the SESYS program should have a way of remembering answers to previous questions. This makes using the program very efficient.

Assume that we have statistical method \textit{paasche index}. When the program starts, the goal \texttt{go_ahead} unifies with the first \texttt{go_ahead} clause and eventually with the statistical rule head, \texttt{statistical_technique(start_program, laspayres_index)}. The test on this statistical rule begins with the first question asking whether we know the current year prices. When we respond \textquote{n}, Turbo Prolog backtracks, and then tries the next statistical rule, headed \texttt{statistical_technique(start_program, laspayres_index)}. Again, Turbo Prolog tries to prove the first premise, \texttt{known(start_program, base_year_prices)}, and asks whether we know the basic prices. The answer again is \textquote{n}, the statistical rule fails, and Turbo Prolog tries the statistical rule \texttt{statistical_technique(start_program, weighted_aggregates_index)}. This process continues until Turbo Prolog reaches the statistical rule \texttt{statistical_technique(start_program, paasche_index)}.

This time the first premise is \texttt{known(start_program, current_year_quantities)}, and the first question asks whether we know the \textit{current year quantity}. As the SESYS moved through the various statistical
rules, Turbo Prolog asked the same questions if they were relevant to the statistical rule being tested because the program could not remember the answer to a question from one rule to the next. What the program needs is some type of database in which to store statistical facts about particular problem during the consultation session.

During the SESYS program execution, statistical facts obtained from the user during the consultation session that apply only to the current consultation are stored in the dynamic database. This is done by creating one or more database predicates and defining these as database predicates in a database session of the program (Townsend, 1988). These database predicates are then updated using one of Turbo Prolog’s built-in database predicates.

With Turbo Prolog, the dynamic database is stored in memory with the static database. Our computer system must contain enough memory to store both databases. We can save the dynamic database to disk (using the save predicate) and read it from disk (using the consult predicate), but during execution it resides in memory with the SESYS program.

Typing Statistical Dynamic Database Predicates. To store information in a dynamic database, we must create one or more database predicates. Statistical facts can then be stored in these predicates during program execution using built-in asserta(fact) and assertz(fact) predicates or by reading facts from a disk file using the built-in consult(filename) predicate.

To create database predicates, we must add a database section to the SESYS program and define the predicates in the predicates in this section. This section must follow the domains section and precede the predicates section. The SESYS with a dynamic database added is shown in Figure 4.2.
domains
  technique, known = symbol
  query = string
  reply = char

database
  xpositive(known)
  xnegative(known)

predicates
  statistical_technique(technique)
  known(known)
  response(reply)
  goAhead
  positive(query, known)
  clear_facts
  remember(known, reply)
  ask(query, known, reply)

clauses
  goAhead :-
    clearWindow,
    statistical_technique(Technique), !,
    write(Stat_problem, "needs the ", Technique, " to solve your problem"),
    clear_facts.

  goAhead :-
    write("Sorry, I am unable to find out the statistical technique"),
    write(" required to solve your problem."),
    write(" Better consult a statistician"),
    clear_facts.

  positive(_, Known) :-
    xpositive(Known), !.

  positive(Query, Known) :-
    not(xnegative(Known)),
    ask(Query, Known, Reply),
    Reply='y'.

  ask(Query, Known, Reply) :-
    write(Query), nl,
    readchar(Reply),
    write(Reply), nl,
    remember(Known, Reply).

  remember(Known, 'y') :-
    asserta(xpositive(Known)).

  remember(Known, 'n') :-
    asserta(xnegative(Known)).

  clear_facts :-
    retract(xpositive(_)), fail.

  clear_facts :-
    retract(xnegative(_)), fail.
clear_facts.

known(current_year_prices) :-
    positive("Do you know the current year prices (y/n)?", current_year_prices).

known(base_year_prices) :-
    write("Do you know the base year prices (y/n)?", base_year_prices).

known(current_year_quantity) :-
    write("Do you know the current year quantity (y/n)?", current_year_quantity).

known(base_year_quantity) :-
    write("Do you know the base year quantity (y/n)?", base_year_quantity).

known(quantities_from_a_representative_year) :-
    write("Do you know the quantities from a representative year (y/n)?", quantities_from_a_representative_year).

known(no_of_elements_used_in_index) :-
    write("Do you know the number of elements used in Index calculation (y/n)?", no_of_elements_used_in_index).

known(quantities_prices_that_determine_values_we_use_for_weights) :-
    write("Do you know the quantities and prices that determine values we use for weights (y/n)?", quantities_prices_that_determine_values_we_use_for_weights).

statistical_technique(unsigned_aggregates_index) :-
    known(current_year_quantity),
    known(base_year_quantity).

statistical_technique(weighted_aggregates_index) :-
    known(current_year_prices),
    known(current_year_quantity),
    known(base_year_prices).

statistical_technique(laspeyres_index) :-
    known(current_year_prices),
    known(base_year_prices),
    known(base_year_quantity).

statistical_technique(paasche_index) :-
    known(current_year_prices),
    known(base_year_prices),
    known(current_year_quantity).

statistical_technique(fixed_weight_aggregates_prices_index) :-
    known(current_year_prices),
    known(base_year_prices),
    known(quantities_from_a_representative_year).

statistical_technique(unweighted_average_of_relatives_prices_index) :-
    known(current_year_prices),
    known(base_year_prices),
known(no_of_elements_used_in_index).

statistical_technique(weighted_average_of_relatives_priceindex):-
    known(current_year_prices),
    known(base_year_prices),
    known(quantities_prices_that_determine_values_we_use_for_weights).

statistical_technique(weighted_average_of_relatives_quantityindex)
    :-
    known(base_year_quantity),
    known(current_year_quantity),
    known(quantities_prices_that_determine_values_we_use_for_weights).

response(Reply) :-
    readchar(Reply),
    write(Reply), nl.

/* End of Program */

Figure 4.2: Dynamic Database in SESYS

The number of database predicates defined depends upon the application (Townsend, 1988). The SESYS uses two database predicates. One is for statistical facts proven true, and the other is for statistical facts proven false.

\[
\text{positive}(\text{known}) \\
\text{negative}(\text{known})
\]

In the SESYS, the \text{positive}(\text{known}) predicate is used to store statistical facts proven true, and the \text{negative}(\text{known}) predicate is used to store statistical facts proven false. With these we can store statistical facts learned from the questions in the database and query the database before asking the same question again.

**Operations on the SESYS Dynamic Database.** The database predicates can be used like any other predicate. They are considered true if a tally is found in the database and false if no tally is found. New statistical facts are stored in database predicates using the \text{asserta}(\text{facts}) and \text{assertz}(\text{fact}) built-in predicates. Statistical facts are removed from the predicate using \text{retract}(\text{fact}).
The go ahead clause invokes `statistical_technique(Technique)`. The first premise of the statistical rule to test is `known(current_year_prices)`. The first premise in the `known(current_year_prices)` rule is `positive(query, known)`. The first clause unifying with the `positive` predicate is

```
positive(_, Known) :-
    xpositive(Known), !.
```

The first `positive` rule checks the database for `xpositive(current_year_prices)` (we know the current year prices), this tally will fail, because the database currently is empty. The program then backtracks to

```
positive(Query, Known) :-
    not(xnegative(Known)),
    ask(Query, Known, Reply),
    Reply='y'.
```

The second `positive` clause checks the other statistical database predicate for `xnegative(current_year_prices)` (current year prices are not known). This tally fails, but the failure is negated with the 'not' predicate. The `ask(Query, Known, Reply)` predicate ask the question and invokes `remember(Known, 'y')` or `remember(Known, 'n')`. These statistical rules store the answer to one or the other of the statistical database predicates using the `asserta(fact)` built-in predicate.

The purpose of the cut in the first `positive(_, Known)` clause is to prevent backtracking and bind the program to this clause if it succeeds. For example, if we know only the current year prices but not the current year quantity, the first `statistical_technique(unsigned_aggregatesindex)` rule would fail, causing Turbo Prolog to backtrack to `statistical_technique(weighted_aggregatesindex)`. We know the `current_year_prices`, so the first `positive(_, Known)` clause succeeds. Now, if any further premise of `statistical_technique(weighted_aggregatesindex)` fails, the cut prevents the program from backtracking and typing the second `positive(Query, Known)` clause.
Here are some general rules for storing the statistical facts into a statistical database:

1. The `asserta(fact)` predicate stores a statistical fact at the beginning of the statistical database. The `assertz(fact)` predicate stores a statistical fact at the end of the statistical database. Which of these we use depends on where we wish to put the statistical fact in the statistical database. Choose location according to the frequency with which the fact will be accessed to create a more efficient program.

2. All variables in fact must be bound before the statistical database predicate -- `asserta(fact)` or `assertz(fact)` -- is invoked.

**Saving the SESYS Dynamic Database.** In the SESYS system, the statistical database applies only to one consultation session and is cleared before the program is restarted. In some cases, however, we may wish to save the dynamic statistical database and use it again. To save the current dynamic statistical database, we use the built-in `save(filename)` predicate, specifying the file name and extension as follows:

```
save("SMETHOD.DAT")
```

The predicate always evaluates as true. The resulting file is a collection of statistical facts and looks like any Prolog program except that it does not include section names.

At any time later in this program (or another program), you may read this dynamic statistical database into the program and use it with the `consult(filename)` built-in predicate as follows:

```
consult("SMETHOD.DAT")
```

Also, this predicate always evaluates as true. Once the statistical facts are read into memory, they are part of the current program.
Removing Statistical Facts from the SESYS Database. Once a statistical fact is stored in the statistical database, it can be removed only using the built-in \texttt{retract(fact)} predicate. The \texttt{retract(fact)} predicate removes statistical fact from the statistical database. For example,

\begin{verbatim}
retract(xpositive(current_year_prices))
\end{verbatim}

removes the statistical fact \textit{current year prices} from the statistical database \texttt{xpositive(fact)}.

Variables do not have to be bound before the predicate is invoked. For example,

\begin{verbatim}
retract(xpositive(_))
\end{verbatim}

removes the first statistical fact in the statistical database tallied with the specified condition.

In the SESYS system, the dynamic statistical database must be cleared before the information on the next statistical method is entered. This is done with the \texttt{clear facts} predicate, as follows:

\begin{verbatim}
clear_facts :-
    retract(xpositive(_)), fail.

clear_facts :-
    retract(xpositive(_)), fail.
clear_facts.
\end{verbatim}

This code includes two failure loops. The first clause,

\begin{verbatim}
clear_facts. :-
    retract(xpositive(_)), fail.
\end{verbatim}

removes one item from the statistical database and fails at the predicate. This forces backtracking to the \texttt{retract} predicate, which removes another item from the statistical database. Eventually, the statistical database is empty, and Turbo Prolog backtracks to the second clause,

\begin{verbatim}
clear_facts. :-
    retract(xpositive(_)), fail.
\end{verbatim}
The second clause loops in the same way, clearing the second statistical database predicate. Like the first clause, the retract predicate eventually fails here, forcing backtracking to the third clause, which always succeeds.

To modify or update a statistical fact in the statistical database, we must first delete it and then add it again with the changes incorporated. There is no built-in predicate for editing a statistical fact in the statistical database.

4.4 Interfacing the SESYS with “C” Program

This section describes the ways in which Turbo Prolog interacts with programs and routines written in “C” language, with the DOS operating system, and with statistical packages like SPSS, STATISTICA, and SYSTAT.

Although Turbo Prolog is a powerful and flexible language, it is not suitable to all programming tasks. Having understood the concept of modular programming and capabilities of the “C” language, we want to design the SESYS in which some of the processing is handled outside of Turbo Prolog, with the results available to Turbo Prolog program.

We must be aware of the design issues when we write external procedures in ‘C’ that are called from SESYS:

- Declaration of the ‘C’ routine
- Appropriate naming procedure in ‘C’ routine
- Proper use of parameters
- Selection of the large memory model

We want to device an interaction between Turbo Prolog and ‘C’ language when existing data and programs are more usable. For example, we may need to interact with spreadsheet that calculates cash flow position or we may need those results from the same disk where Turbo Prolog and the programs are stored.
We may want to undertake some actions in our Turbo Prolog programs that require more direct access to DOS and the internals of the PC than is permitted by Turbo Prolog's high-level language design. This "bit-bidding" or "bit-fitting" might involve real-time data collection, communications port management, direct memory management, or other activities that take place at a far lower level than that on which Turbo Prolog operates.

There are two ways an interaction can take place between Turbo Prolog and 'C' language. These are summarised in Table 4.1. The direct call interaction is discussed in greater detail in the following sections.

**Direct Call Method.** Turbo Prolog includes built-in predicates and declarations that permit a Turbo Prolog program to call directly a routine.

**Table 4.1: Basic Methods of Interaction**

<table>
<thead>
<tr>
<th>Type</th>
<th>Interaction Through</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate file</td>
<td>A file accessed by both programs</td>
<td>Less than automatic, not efficient.</td>
</tr>
<tr>
<td>Direct call</td>
<td>Predeclared Turbo Prolog external Predicates</td>
<td>Difficult to program. Both programs need special care.</td>
</tr>
</tbody>
</table>


written in 'C' language. This method is much more direct and automatic than the intermediate file approach.

Figure 4.3 depicts a typical direct call flow diagram sequence involving a Laspeyers Index method routine written in 'C' language. Interfacing with 'C' routine written for Laspeyers Index method will be discussed later in greater detail.
While writing the ‘C’ routine, file handling concept of ‘C’ is used. The processing of the direct call method of program interaction is straightforward and easy to understand. It does, however, require careful coding in the Turbo Prolog making the call.

**Flow of Direct Call Interaction with 'C'.** When Turbo Prolog interacts with ‘C’ routine through the direct call technique, there are a number of rules to follow:

1. A global predicate must be declared. The Turbo Prolog program must include a global predicates segment that declares one or more predicates to be external to the program and explicitly identifies the ‘C’ language.

![Diagram of SESYS Direct Call Interaction with 'C'

Figure 4.3: SESYS Direct Call Interaction with 'C'
2. Flow control pattern(s) must be included. In the global predicate segment, the flow pattern(s) of the predicate must be explicitly provided.

3. The predicate must be used in a global declaration or statement. The predicate declared as external is then used exactly as if it were a Turbo Prolog goal or statement. It is simply stated, with the appropriate number of parameters, in the appropriate place in the program.

4. The external program name must follow a convention. Because of the possibility of alternate or compound flow patterns in global predicates, Turbo Prolog automatically assigns an appendage to the predicate name. This appendage must be explicitly included in the name of the 'C' language routine being called (Shafer, 1987).

Each of the rules is defined below for Laspeyres index method.

**Flow Pattern Declaration.**

```
global predicates
laspeyresindex(real, real, real, real) - (i,i,i,o) language C
```

The first part of this declaration is a predicate named `laspeyresindex` that is associated with four parameters, all of which are real numbers. Following this otherwise ordinary predicate declaration is a hyphen followed by the expression `(i,i,i,o)`. This is the flow pattern for this predicate. Any flow pattern description in Turbo Prolog consists of two or more letters enclosed in parentheses and separated by commas. The two allowable letters are `i` and `o`. `i` stands for input and `o` for output. It indicates that the first three real numbers following the `laspeyresindex` predicate are known at the time the 'C' routine call is made and that the fourth is unknown. In other words, if all values are variables, the first three are instantiated before the predicate is called and the fourth is not.

**The SESYS Global Domain and Predicate Declarations.** To enable more than one module of the SESYS to use the same predicate and its
associated domain values, predicates and domains must be explicitly declared as global.

```prolog
global domains
  value=string
  valist=integer*
```

After a domain is declared global in this way, its value is available to all modules in SESYS. This includes programs written in ‘C’ language or in languages other than Turbo Prolog (Shafer, 1987).

**Declaring the ‘C’ Language.** All external calls in the SESYS do not need to be written in the same language. Each predicate must have its own language declaration in any event, so mixing languages such as ‘C’, Pascal, FORTRAN, and Assembly is valid.

In most external calls to routines, written in ‘C’ language, at least one argument has a flow pattern of 0, indicating it is unknown when the routine is called. If this were not the case, the Turbo Prolog code and the external program would not have a common variable to store and retrieve the result of a routine. In other words, there would be no vehicle by which the two languages could interact.

**Proper Use of Parameters.** If the parameter is passed from the SESYS to ‘C’ routine, it is pushed onto the stack as a 32-bit pointer to the memory location where the return value must be stored before the control returns to the SESYS. Input parameters passed to the SESYS are pushed directly onto the stack; and the size of the parameter on the stack depends on the data type. Refer to Turbo Prolog Users' Manual for more details. Turbo Prolog requires that all pointers be 32-bit values. All these issues are automatically handled by ‘C’.

**Selection of Large Memory Model.** Many high level languages, including ‘C’, allow us to choose a large or small memory model when compiling and linking programs. The SESYS requires that we always use the
large memory model when programming ‘C’ routines, if the option is available.

4.5 The ‘C’ Routine

This section provides a useful ‘C’ routine and shows how it can be implemented and called from Turbo Prolog.

This small routine is written in Microsoft C, Version 6.0 (Radcliffe, 1996). It computes the laspeyres index for management, and returns the answer as real. The input data are read from an user-defined input file (Figure 4.4).

/* Begin of Program */

```c
int_acrtused;
laspeyresindex(name)
    char *name
    { 
      FILE *fp;
      int err,size_float, number_records;
      float tmp_p1, tmp_p2, tmp_q0;
      float sum_plq0,tmp_plq0
          sum_plq0,tmp_plq0, last_index;
      size_float = sizeof(float)
     fp = fopen("record","r");
     if (fp ===NULL)
       printf("Error unable to open file\n");
       fseek(fp,20,seek_set);
     
     while(1)
          { 
            fseek(fp,7,seek_cur);
            err = fread(&tmp_p1, size_float,1,fp)
            if(err != 1)
               { 
                printf("End of record\n");
                break;
               }
            
            fseek(fp,3,seek_cur);
            err = fread(&tmp_p2, size_float,1,fp)
            if(err != 1)
               { 
                printf("End of record\n");
                break;
               }
            
            fseek(fp,3,seek_cur);
            err = fread(&tmp_q0, size_float,1,fp)
            if(err != 1)
               { 
                printf("End of record\n");
                break;
               }
```
```c
		tmp_plq0 = tmp_pl * tmp_q0;
	sum_plq0 += tmp_plq0;
	tmp_p2q0 = tmp_p2 * tmp_q0;
	sum_p2q0 += tmp_p2q0;
	number_records++;
```}

```c
printf("Number of Records Read : %d\n", number_records);
lasp_index = (sum_plq0 / sum_p2q0) * 100;
printf("Laspeyres Index Value = %f", lasp_index);
return(lasp_index);
}

/* End of Program */
```

Figure 4.4: ‘C’ Routine to Compute Laspeyres Index Value

Note that the strange looking line at the beginning of this program.

```c
int_acrtused;
```

This statement prevents a link-error. It is required only if we prepare
this file for the Microsoft ‘C’ compiler. We enter this program and save it as
LASPEYRES.C.

4.6 A Part of the SESYS Program

The SESYS Program uses the ‘C’ program we have just saved as
LASPEYRES.C. This program is a very small segment of the SESYS
(Figure 4.5). But this small routine serves the purpose very well.

```prolog
/* Begin of Program */
global predicates
   _laspeyresindex(string,real,real,real,real) - (i,i,i,i,o)language C

predicates
   lasp_index(real)
goal
   lasp_index(Value)
clauses
   lasp_index(Indexvalue) :-
      clearwindow,
      write("Enter the file name:");
      readln(Filename),
      _laspeyresindex(Filename,P1, P2, Q0, Value),
      write("The Laspeyres Index value is ",Value).

/* End of Program */
```

Figure 4.5: Calling Laspeyres Index ‘C’ Routine Through Turbo Prolog
Notice that in both Turbo Prolog and ‘C’ routine, an underscore character is placed in front of the name of the ‘C’ predicate to be called. This is required by Microsoft ‘C’ s methods of calling external routines. If we are using a different ‘C’ compiler, this may not be necessary, but including it will not cause a problem.

We type this program into Turbo Prolog and save it as LASPINDEX.PRO. We will use this name in further processing steps.

In the next section, we shall see how the ‘C’ routine and the SESYS programs are compiled individually and linked together for execution.

**Compiling the ‘C’ Program (LASPEYRES.C).** Compile the ‘C’ program LASPEYRES.C so that it can be linked with the SESYS. We use the following compilation sequence:

```
msc /Gs /AL LASPEYRES.C
```

The ‘Gs’ option turns off stack checking. The ‘AL’ argument results in the use of the large memory model, as required by the SESYS. When this compilation process is complete, a file called LASPEYRES.OBJ exists on the hard disk.

**Compiling the Prolog Program (LASPINDEX.PRO).** Next, we compile LASPINDEX.PRO as an executable file using the built-in Turbo Prolog compiler. First, we use the **Options selection** from the **Compile menu** to choose a file of type .EXE. Then we choose the **Compile option**. This creates a new object file, LASPINDEX.EXE.

**Linking the Two Programs.** Now let us create the SESYS project file with the same name as the main file it contains. The new file, SESYS.PRJ, contains this line:

```
LASPINDEX + LASPEYRES +
```

After we create the project file, select **Compile Project** under the **Options** pull-down menu, give Turbo Prolog the name of the project file, and choose the Compile option.
4.7 Calling External Statistical Packages from the SESYS

In the previous section we looked at how to interface the SESYS with programs written in 'C' language. In this section, the focus is on how to call the programs such as the SPSS, and STATISTICA and not to return some value to the SESYS but rather to perform some external functions.

The Turbo Prolog built-in predicate system can call any program stored external to Turbo Prolog as a batch or command file; that is, any file with the extension of .BAT, .EXE, or .COM and the use of the predicate is straightforward. Suppose we have a file called SPSS.EXE or STATISTICA.EXE on the logged disk drive, and we want to execute it within the SESYS program. We could simply code a line as follows:

```prolog
system(spss). or
system(statistica).
```

On encountering that line, the SESYS goes to the hard disk, uses the current DOS path to locate the file, and executes it. When the SPSS or STATISTICA package finishes executing, the control returns immediately to the SESYS. This last point is important. If we are calling an external program that displays information on the screen (such as a specialised directory routine), we may have to modify the external program to pause when it is finished and wait for the user to press a key to return to the SESYS. Otherwise, the displayed information disappears so rapidly that it is useless.

One way to do this is to use the redirection capabilities of DOS and route the output through the DOS MORE filter. For example, we could execute the file OUTPUT1.COM by setting up a batch file called OUTPUT2.BAT. In this file, we would have just one line:

```
Output1| more.
```

Output1| more. The program will pause at each full screen and before returning control to the SESYS. If the external program we are calling returns some value to SESYS and if we do not want to link the program
directly into your code using the techniques described in the previous section, then we should pass the information by writing it to a file.

There is always the feasibility to use SESYS in an on-line context and data processing can be accomplished through statistical analysis on-line and interpretation thereof, towards information management.

4.8 How SESYS Works Towards Information Management

At this juncture, it becomes necessary to properly describe how exactly information management can be accomplished by the use of SESYS, by novice and expert users of statistics, whether as individuals as personnel of the Corporate industries and various other agencies, government or otherwise. Figure 4.6 shows the route that helps the SESYS user towards information management. There are seven steps to information management through interpretation of the data analysed using select methods suggested by the SESYS. They (1-7) are identified as processes (boxes with thick borders), through actions/activities (a-g, seven steps), the last being both a process and action/activity. Search for data sources lead the user of SESYS on to some data source (step 1), which can be tapped for data using appropriate collection methods (step 2). Once the data is collected, a method or methods of data analysis is/are selected using the SESYS (step 3), which in the actual process helps transform the data (step 4) and analyse data by the SESYS, or SESYS-given methods (step 5). In step 6, the processed data, that is, information is interpreted such that the interpretations or results of that process are used in information management in the organisation/institution where the personnel using the SESYS works. As step 7 indicates that the information management leads on to management solutions to the Corporate industry or the private individuals using the SESYS. Each of the 7 steps is preceded or accompanied by an action/activity which the user(s) of the SESYS perform such that information management is complete. Collection of data and selection of statistical method is preceded by use of select
methods of collection (step a), followed quickly by data acquisition (step b), and facilitation of data analysis (step c).

Figure 4.6: Processes and Actions/Activities and their sequences towards Information Management and Management Solutions

Then follows the activity of interfacing SESYS and accessing packages for data processing or on-line accessing and processing of data (step d). The interpretations of the data analysed, that is, information leads the information manager in the user(s) of SESYS to decide on a course of action (step e), which ultimately ends up in application / decision-making.
4.9 SESYS Model Runs: KBSSP and SESA

Knowledge Based Systems and Selection of Methods. The Knowledge-Based Statistical Software Program (KBSSP) is an attempt to develop a statistical knowledge-based expert system software, which may be used to lead to better statistical significance tests, analysis of variance, multivariate and factor analysis.

During the development of the KBSSP, attention has been focused on the necessary modules of the interface between the user and the modules, which give actual consultation. The KBSSP should offer a simple and functional user interface, which only uses function keys F1-F5 to provide access to its sub-menu utilities (help, library, glossary, input). The space bar and the <Enter> key are to continue or acknowledge. The numeric keypad, the Y/y key or the N/n keys are for answering the queries. The user can scroll the menus or navigate through the embedded menus of the glossary with the arrow keys (or the PgUp and PgDn keys).

In addition, the KBSSP should offer the following significant utilities (available with a single key depression) that support the users taking the advice:

1. The capability to exit at any time and to restart from that point the next time;
2. A library containing information on other or the same methods; and
3. Online help for the system functions or a list of suggested methods for administrators and managers to read or to consult the real statistician in the case of complex environment.

The basic structure selected for the development of the KBSSP will be a modular one. Further, the structure of the expert module of the KBSSP should follow cognitive modelling approach, where knowledge is
decomposed into meaningful, integrated components, and thus it may be easily understood and applied. The separation of the KBSSP into modules will reduce the complexity of the overall system design and facilitates easy modification and enhancement of the system and also its related debugging activities.

One module will be mainly developed for consultation session. The consultation session will consist of a series of questions, which the computer asks the users in order to determine the exact nature of the statistical problem of interest and therefore the appropriate statistical procedure for use. Once all the questions have been answered by the users, the computer responds either by displaying statistical method it recommends along with a short write-up on computational procedures or it suggests consultation with a ‘real’ statistical expert in case the problem is complex or unique. Once this session is over, the KBSSP will prompt the user to continue on to the next session.

The knowledge base would contain knowledge about the domain of statistics. The KBSSP system will offer advice on the following topics:

1. Comparison of two groups of data;
2. Comparison of two groups on one or more dimensions;
3. Analysis of repeated measures design;
4. Comparing the groups with respect to categorical outcome measures; and
5. Fitting models of relationships amongst variables.

Consequently, the rule-base module will contain, for example, 20 rule heads to propose 20 methods. The modular design of the KBSSP enables new knowledge / method is easily added or modified and when the KBSSP is increasingly used, its expertise will be improved accordingly.
The salient features of the KBSSP are:

1. The KBSSP is very easy to use. Therefore, there is no need for the users to be trained in using the KBS tool; its simple interface, together with the on-line help, makes it easy for users to use. The users are not computer experts.

2. The users get the feeling very early on that they are in control of the process with no additional effort.

3. The KBSSP will be autonomous, complete with respect to the advice/taught methodology, portable, user-controlled and easy-to-use.
A Sample Terminal Session with the KBSSP. To start the sample session, let us assume that psychologists would like to know the statistical tools available to the psychologists working in the area of psychiatry for the following problem:

psychologist wants to study the length of stay of certain patients in a hospital. On treatment, each patient is given a certain type of medicine and a certain type of vitamin supplement. The psychologist wishes to test whether there are differences in the average length of hospital stay that can be attributed to the type of medicine given or to the type of vitamin supplement given. And also he wishes to determine if the mean length of stay in the hospital for patients are related to the type of medicine and type of vitamin supplement that are administered. Three types of vitamin supplements (V₁, V₂ and V₃) and three types of medicine (M₁, M₂ and M₃). Total number of patients (constant c) is 4. Four patients were given the combination (M₁, V₁), four were given the combination (M₁, V₂), and so forth. The data are given in table 4.7. Use a 5% level of significance and test the three null hypotheses that the medicine effects, the vitamin effects, and the interaction effects are 0.

<table>
<thead>
<tr>
<th>Vitamin Supplement</th>
<th>M₁</th>
<th>M₂</th>
<th>M₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>V₁</td>
<td>8</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>V₂</td>
<td>9</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>V₃</td>
<td>4</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>
From the problem above, we can understand that the following questions are to be answered:

a) The appropriate statistical technique needed to tackle the above problem.

b) Calculate the sample mean for each cell.

c) Calculate the sample mean for each brand of vitamin.

d) Calculate the sample mean for each type of medicine.

e) Denote brand of vitamin as Factor A and type of medicine as Factor B & Calculate Sum of Squares of Factor A (SSA), Sum of Squares of Factor B (SSB), Sum of Squares of Interaction (SSI), Sum of Squares of Error (SSE) and Sum of Squares of Total (SST).

f) Calculate the mean squares of factor A, B and Error which are MSA, MSB and MSE.

g) Calculate the $F$ statistic (Ratio) $F = \frac{MSI}{MSE}$ and test that the interaction effects are 0 using $\alpha = 0.05$.

**Solution.** The user is interested in determining the statistical method(s) and the solution to the problem. To solve this problem, the user has to access the KBSSP. The introductory module is executed first by displaying the statistical areas covered by the software. The users are directed to seek advice only from the areas specified in the introductory panel (Figure 4.8). The second sub-module, probably the rule-base module, is then executed by initiating the query phase of the session. The sequence of questions initiated by the SESYS and the user’s response (through dialogue) for accessing the statistical technique is shown in Figure 4.9. As the query session runs into number of screens, only the portion relevant to our problem is shown in the dialogue screen.
You are asked to limit yourself to obtain advice on statistics only on the following topics:

1. Comparing two groups.
2. Comparing two groups on one or more dimensions.
4. Comparing groups with respect to categorical outcome measures.
5. Fitting models of relationship amongst variables.
6. Hypothesis testing.
7. Chi-Square tests.
8. Analysis of variance.
9. Regression and Correlation.
10. Multiple regression Models.
11. Estimation of the trend component.
12. Estimation of the seasonal component.
13. Some non-parametric tests.
14. Index numbers

By inference the KBSSP determines the statistical technique/method of Two-Way analysis of variance (Two-Way ANOVA) with interaction effects which meet the users' requirements and proposals. The statistical technique suggested by the KBSSP for the psychologist with step-by-step instructions and computational algorithm highlighted in the computer screens are shown in Figure 4.10.
To end SESYS dialogue press a * and Enter otherwise just press Enter key ...

- Are you interested in fitting models of relationship amongst variables? N
- Are you interested in comparing variables towards categorical outcome measures? N
- Do you want to use the analysis of repeated measures designs? N
- Do you want to use correlation analysis? N
- Do you want to use tests for differences between various conditions of variables? Y
- How many independent variables do you have? 2
- Is this variable associated with any other variable? Y
- Does this variable depend on another variable, for decision making? Y
- Whether the same (or matched) variable is used for all experimental conditions? Y
- Whether each possible combination of values is called a treatment? Y
- Do you have exactly one observation per cell in your problem? N
- Do you have a problem which contains multiple observations per cell? Y
- Do you want to estimate and test for the presence of interaction effects between the two factors? Y
- Do the model relates the value of the dependent variable to two independent variables and to random effects? Y
- Each observation belongs to one of \( k \) groups or populations based on the value of a single factor? Y
- Each observation is classified according to the values of two independent variables or factors? Y
- Do you have ‘\( a \)’ different levels for factor A and ‘\( b \)’ different levels for factor B (where, \( a \) and \( b \) are constants)? Y
- Are there \( (axb) \) different possible combinations of values for the two factors? Y
- Whether the treatments effects are not the same for different blocks/various types/groups? Y
- Are you interested to find out \( F \) Ratio? Y
- *

Press: <F1> for Help  \(<\text{Enter}>\) for continue  \(<\text{Esc}>\) for Abort

**Figure 4.9:** Dialogue for Accessing the Statistical Method (KBSSP)
The suggested statistical technique and technical procedures for the proposed problem by the Psychologist is given as follows:

**Two-way ANOVA Model with Interaction Effects**

The general two-way ANOVA model with interaction effect can be expressed as

\[ X_{ijk} = \mu + A_i + B_j + I_{ij} + e_{ijk} \quad \text{where} \quad i=1,2,\ldots,a; j=1,2,\ldots,b; k=1,2,\ldots,c \]

Arrangement of data for a two-way ANOVA with multiple observation per cell can be arranged as follows:

<table>
<thead>
<tr>
<th>Level of Factor B</th>
<th>1</th>
<th>2</th>
<th>\ldots</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of Factor A</td>
<td>( X_{111} )</td>
<td>( X_{121} )</td>
<td>\ldots</td>
<td>( X_{1b1} )</td>
</tr>
<tr>
<td></td>
<td>( X_{112} )</td>
<td>( X_{122} )</td>
<td>\ldots</td>
<td>( X_{1b2} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( X_{11c} )</td>
<td>( X_{12c} )</td>
<td>\ldots</td>
<td>( X_{1bc} )</td>
</tr>
</tbody>
</table>
| \hline
| Level of Factor A | \( X_{211} \) | \( X_{221} \) | \ldots | \( X_{2b1} \) |
|                   | \( X_{212} \) | \( X_{222} \) | \ldots | \( X_{2b2} \) |
|                   |         |         |        |       |
|                   | \( X_{21c} \) | \( X_{22c} \) | \ldots | \( X_{2bc} \) |
| \hline
| Level of Factor A | \( X_{a11} \) | \( X_{a21} \) | \ldots | \( X_{ab1} \) |
|                   | \( X_{a12} \) | \( X_{a22} \) | \ldots | \( X_{ab2} \) |
|                   |         |         |        |       |
|                   | \( X_{a1c} \) | \( X_{a2c} \) | \ldots | \( X_{abc} \) |
| \hline

Press: <F1> for Help <Enter> for continue <Esc> for Abort

Figure 4.10: The SESYS's Recommendation for Two-Way ANOVA with Interaction Effects.
Thus, the required method or statistical advice to study the length of stay of certain patients desired by the psychologist is highlighted by the KBSSP. In case the user fails to elicit any more statistical advice or to learn, he/she may invoke the KBSSP once again from the introductory panel.

In this model

i) The random variable \( X_{ijk} \) is the \( k^{th} \) observation associated with level \( i \) of the row factor (Factor A) and level \( j \) of the column factor (Factor B).

ii) the total combinations of the levels of Factor A and Factor B is \( ab \).

iii) Total observations for each combination is \( c \) (where, \( c \) is a constant).

iv) The value \( X_{ijk} \) is the \( k^{th} \) observation in cell \( (i,j) \).

v) \( x_{ijk} \) denotes the observed value of the random variable \( X_{ijk} \).

vi) The random effect is \( e_{ijk} \).

vii) \( A_i \) represents the effect of the \( i^{th} \) level of Factor A and \( B_j \) represents the effect of level \( j^{th} \) of Factor B.

viii) \( l_{ij} \) represents an interaction effect between level \( i \) of Factor A and level \( j \) of Factor B.

ix) Each cell contains \( c \) observations, so the total number of observations is \( n=abc \).
The following three null hypotheses that can be tested:

1. No difference in the population means associated with different levels of Factor A.
2. No difference in the population means associated with different levels of Factor B.
3. No Factor A-Factor B interaction.

To prefer the above tests, it is necessary to calculate various sample means and sums of squares.

They are

i) \( \mu_{ij} \) denote the population mean of all observations at the \( i^{th} \) level of Factor A and the \( j^{th} \) level of Factor B.

ii) \( \mu_i \) denote the population mean of all observations at the \( i^{th} \) level of Factor A.

iii) \( \mu_j \) denote the population mean of all observations at the \( j^{th} \) level of Factor B.

iv) \( \bar{x}_{ij} \), \( \bar{x}_i \) and \( \bar{x}_j \), denote the corresponding sample means.

Note: The location of the dot in the subscript indicates the index over which the sum is taken.

The effects (\( A_i \), \( B_j \), interaction effect \( I_{ij} \) and random effect \( e_{ijk} \)) are computed as

\[
A_i = \mu_i - \mu \\
B_j = \mu_j - \mu \\
I_{ij} = \mu_{ij} - \mu_i - \mu_j + \mu \\
e_{ijk} = X_{ijk} - \mu_{ij}
\]
Enter the data as proposed by the SESYS and the table will be displayed as follows:

<table>
<thead>
<tr>
<th>Vitamin Supplement</th>
<th>M₁</th>
<th>M₂</th>
<th>M₃</th>
<th>( \bar{x}_{ij} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>V₁</td>
<td>8</td>
<td>5</td>
<td>12</td>
<td>( \bar{x}_{11} = 7 )</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>10</td>
<td>( \bar{x}_{12} = 6 )</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>( \bar{x}_{13} = 11 )</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>( \bar{x}_{1} = 8 )</td>
</tr>
<tr>
<td>V₂</td>
<td>9</td>
<td>14</td>
<td>10</td>
<td>( \bar{x}_{21} = 9 )</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>( \bar{x}_{22} = 10 )</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>8</td>
<td>( \bar{x}_{23} = 8 )</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>( \bar{x}_{2} = 9 )</td>
</tr>
<tr>
<td>V₃</td>
<td>4</td>
<td>15</td>
<td>11</td>
<td>( \bar{x}_{31} = 5 )</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>13</td>
<td>12</td>
<td>( \bar{x}_{32} = 14 )</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16</td>
<td>13</td>
<td>( \bar{x}_{33} = 11 )</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>8</td>
<td>( \bar{x}_{3} = 10 )</td>
</tr>
</tbody>
</table>

Therefore, the statistical model becomes as:

\[
X_{ijk} = \mu + A_i + B_j + I_k + e_{ijk}
\]

\[
x_{ijk} = (\mu - \mu) + (A_i - \bar{A}) + (B_j - \bar{B}) + (I_k - \bar{I}) + (e_{ijk} - \bar{e})
\]

The corresponding sample identity is

\[
(x_{ijk} - \bar{x}) = (x_{ij} - \bar{x}) + (x_{i} - \bar{x}) + (x_{j} - \bar{x}) + (x_{k} - \bar{x}) + (x_{ijk} - \bar{x}_{ijk})
\]

Press:  <F1> for Help    <Enter> for continue    <Esc> for Abort
If both sides of this equation are squared and summed over \(i, j\) and \(k\), we have

\[
\sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} (x_{ijk} - \bar{x})^2 = \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} ( \bar{x}_{i.} - \bar{x} )^2 + \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} ( \bar{x}_{.j} - \bar{x} )^2 + \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} ( \bar{x}_{..k} - \bar{x} )^2 + \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} ( x_{ijk} - \bar{x}_{ij.} )^2
\]

This equation is the sum of squares identity that forms the basis for the two-way analysis of variance test with interaction.

1. Sum of squares identity with multiple observations per cell.
   
   i) The sum of squares identity is \( \text{SST}=\text{SSA}+\text{SSB}+\text{SSI}+\text{SSE} \)
   
   ii) The corresponding degrees of freedom are
   
   \( (abc-1)=(a-1)+(b-1)+(a-1)(b-1)+ab(c-1) \)
   
   Where the sums of squares and associated degrees of freedom are as follows:

<table>
<thead>
<tr>
<th>Source of variation</th>
<th>Sum of squares</th>
<th>Degrees of freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor A</td>
<td>( \text{SSA}=bc \sum_{i=1}^{a} (\bar{x}_{i.} - \bar{x})^2 )</td>
<td>( a-1 )</td>
</tr>
<tr>
<td>Factor B</td>
<td>( \text{SSB}=ac \sum_{j=1}^{b} (\bar{x}_{.j} - \bar{x})^2 )</td>
<td>( b-1 )</td>
</tr>
<tr>
<td>Interaction</td>
<td>( \text{SSI}= \sum_{i=1}^{a} \sum_{j=1}^{b} ( \bar{x}<em>{ij.} - \bar{x}</em>{i.} - \bar{x}_{.j} + \bar{x} )^2 )</td>
<td>( (a-1)(b-1) )</td>
</tr>
<tr>
<td>Error</td>
<td>( \text{SSE}= \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} (x_{ijk} - \bar{x}_{ij.})^2 )</td>
<td>( ab(c-1) )</td>
</tr>
<tr>
<td>Total</td>
<td>( \text{SST}= \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} (x_{ijk} - \bar{x})^2 )</td>
<td>( abc-1 )</td>
</tr>
</tbody>
</table>

Press:  <F1> for Help  <Enter> for continue  <Esc> for Abort
iii) To compute the mean square, divide each sum of squares by its corresponding degrees of freedom.

Therefore,

\[ MSA = \frac{SSA}{a-1} \]
\[ MSB = \frac{SSB}{b-1} \]
\[ MSI = \frac{SSI}{(a-1)(b-1)} \]
\[ MSE = \frac{SSE}{ab(c-1)} \]

2. Two-way ANOVA with multiple observation per cell.

<table>
<thead>
<tr>
<th>Source of variation</th>
<th>Sum of squares</th>
<th>Degrees of freedom</th>
<th>Mean square</th>
<th>( F ) Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor A</td>
<td>SSA</td>
<td>(a-1)</td>
<td>MSA=SSA/(a-1)</td>
<td>MSA/MSE</td>
</tr>
<tr>
<td>Factor B</td>
<td>SSB</td>
<td>(b-1)</td>
<td>MSB=SSB/(b-1)</td>
<td>MSB/MSE</td>
</tr>
<tr>
<td>Interaction</td>
<td>SSI</td>
<td>(a-1)(b-1)</td>
<td>MSI=SSI/(a-1)(b-1)</td>
<td>MSI/MSE</td>
</tr>
<tr>
<td>Error</td>
<td>SSE</td>
<td>ab(c-1)</td>
<td>MSE=SSE/ab(c-1)</td>
<td>---</td>
</tr>
<tr>
<td>Total</td>
<td>SST</td>
<td>abc-1</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

To solve the problem, user is directed to invoke any one of the statistical packages like MINTAB, SPSS or SYSTAT.

Press:  <F1> for Help  <Enter> for continue  <Esc> for Abort
A Sample Terminal Session with SESA. During the development of the SESA, attention was focused on the necessary modules of the interface between the user and the module, which is the actual consultation. The rule-base module has been developed bearing in mind the order in which the questions are asked so that the user sees some kind of pattern and will have an idea of what sort of questions to expect next.

To start the session, let us assume that the administrators would like to know the relative changes in prices over a period of time in the following data (keeping the quantity consumed in 2001 the same as in 2000 given in Table 4.2).

Table 4.2: Sample Data for Use in the Terminal Session of SESA

<table>
<thead>
<tr>
<th>Commodity</th>
<th>Year 2000</th>
<th>Year 2001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Price</td>
<td>Quantity (in units)</td>
</tr>
<tr>
<td>Rice</td>
<td>9.3</td>
<td>100</td>
</tr>
<tr>
<td>Wheat</td>
<td>6.4</td>
<td>11</td>
</tr>
<tr>
<td>Sugar</td>
<td>5.1</td>
<td>5</td>
</tr>
</tbody>
</table>

Source: A Statistical Expert System for Administrators (Mani, 2000).

Solution: The administrator is interested in determining the statistical method and the solution to the problem. To solve this problem, the administrators has to access the SESA system. The introductory module would be executed first by displaying the statistical areas covered by the system. The administrators are then directed to seek advice only from the areas specified in the introductory panel. The second module rule-base is then executed by initiating the query phase of the session (see Figure 4.10).

The SESA determines the statistical method of "Laspeyre’s base weighted aggregate price index" to meet the Administrator’s requirements
and proposes the method with computational algorithm as shown in Figure 4.11.

After displaying the computing algorithm for the selected method, the SESA displays a dialogue to undertake the required calculations. The data may be entered interactively in the specified format on the screen. To execute the chosen method, Turbo Prolog can be, as indicated before, integrated using other computation-oriented languages such as 'C'.

---

**Figure 4.10: Dialogue for Assessing the Statistical Method**
The method proposed to solve the above statistical problem is as follows:

**LASPEYRE'S BASE WEIGHTED AGGREGATE PRICE INDEX.**

**THE COMPUTING PROCEDURES**

\[ P_{01} = \frac{\text{SIGMA} (p_n \times Q_0)}{\text{SIGMA} (P_0 \times Q_0)} \times 100 \]

where,

- \( P_n \) is current year price of commodities
- \( P_0 \) is base year price of commodities
- \( Q_0 \) is base year consumption

To solve the problem, user is directed to invoke any one of the statistical packages like SPSS, STATISCA or SYSTAT.

Press <F1> for Help <Enter> for Continue <Esc> for Abort

Figure 4.11: SESA's Recommendation for Statistical Forecasting Problem
Potential of SESYS as AI Computer-Assisted Instructor / Tutor.

The potential of SESYS, designed for the purpose of statistical analysis, interpretation and information management, has now been demonstrated with illustrative examples. We may now look at the further potential of the SESYS.

One pertinent area of its use could be in the field of intelligent Computer-Assisted-Instruction / Learning (CAIL). The SESYS could be used as artificial intelligence CAI (AICAI) program or intelligent tutoring system (ITS). Thus it can be used as the AICAI tutor for statistics. CAI/L facilitates individualised instruction/learning in a more effective, faster, and less costly manner than traditional teaching.

The SESYS can be used as an instructor or as a problem-solving program that uses artificial intelligence
techniques to solve difficult problems requiring expertise. The SESYS’ components correspond well with the main components of an intelligent tutoring system in that it has:

1. Expert model(s): the knowledge of the subject matter, that is, statistics;
2. Student model, what the student does or does not understand and how he obtains the solution; and
3. The tutoring model, how to teach.

It must be possible to build into the SESYS the possibility of an instructional procedure, in which a student is given a problem, providing a basis for interpreting his (student’s) problem solving behaviour, and also evaluating his solutions and assisting him in his work. This way, the SESYS can be made to monitor the behaviour of the student, looking for discrepancies from the ideal specification (that is, target problem solving model) and also track faults in the student’s presumed world model or inference procedure, and correct the student by instruction.

However, the instruction places additional requirements on the qualities of expertise. As an instructional expert system, the SESYS should be able to solve statistical problems in multiple ways, allowing for syntactic variations in situation-specific models. It must also model human ways of thinking. In explaining, it must go beyond a trace of how the problem is solved. Above all, it must be able to evaluate the student’s work concerning how he solves the statistical problems given.

The User Agencies and Institutions. It follows from the discussion above that the SESYS could be used in two broad types of agencies and/or institutions:

1. In the Corporate Business Environment (CBE) for information management; and
2. in the University/Research Institutional Environment (U/RIE) for the purpose of instruction and learning in statistics and statistical knowledge.

In either, there are indeed several institutional contexts. Whereas CBE can literally provide for thousands of institutions, the U/RIE provide for some specialised institutions where statistics is an integral part of their existence and work. Wherever finances and marketing are dealt with, for example, on a day-to-day basis, there is definite need for using the SESYS, just as where statistics forms part of the scientific research endeavour. This way the SESYS could be used in knowledge flow and facilitating practices of statistics / statistical knowledge in these institutions (Chappelow et al., 2001). Knowledge has the capability to add value to the organisation or individual.

Similarly, the SESYS could be used by the Government agencies such as the Indian Statistical Institutes, Indian Standard Institutions, and all other developmental agencies, which have the need to use statistics and manage information.

4.10 Summary

This chapter has dealt with the implementation of the SESYS in detail, and problem solving through the SESYS. The user and system interfacing with the external routines written in "C" and statistical packages such as SPSS, STATISTICA and SYSTAT have also been indicated. Dry runs have been made for a session with a ‘C’ routine on Laspeyres index and terminal session with Knowledge Based Statistical System for Psychologists (KBSSP) and Statistical Expert System for Administrators (SESA). In the implementation of the SESYS modules, the use of the Driver module, modifying the SESYS modules, stopping the SESYS program, the SESYS input and output, the SESYS consultation paradigm using WINDOWS have