

1. INTRODUCTION

Before any concept about this project, is discussed, I want to thank to you for your interest in the topic presented and for supporting the project by reading this particular report. This project, as you may have already got a lot of information from the title and the abstract, is about developing an integrated development environment for an expert system shell which is known as SSS (pronounced as 'three-S'). The underlying expert system shell is also a new development and has some interesting properties, as you will see shortly. But, there is an important point to mention about the interface developed in this project. This interface is not just an ordinary user interface. The idea was to minimize the amount of code entered by the user and control the structure of the generated code in a rather strict fashion. With this idea as the main point, two separate sections are implemented. First section is the part that accepts the code entered by the user. But, this is totally different from a 'Editor' like approach. The user just clicks the appropriate buttons and the actual code is then generated by the interface. The second part involved, is the parser. The parser component is mainly used to Type / Syntax check of the generated code, before running it. This component is generated by a YACC-like generator called 'Bison'.

Now, it may be the suitable place to take a step further and see an example code which is to be generated by SSS IDE. The example representing the statement of the problem, is to be given in the following section.

1.1 What is the problem to be solved by SSS System ?

Following code segment, is an example of the code written in SSS expert system shell language that is meant to be generated by the developed user interface. Here, many different and rather strange notations can be seen, but hopefully, everything will become clear shortly.

```
/* 8-queens problem */
?- literalize( global(counter)).
?- literalize( vertical(pos)).
?- literalize( queen_position(h_pos, v_pos)).

context(all).

?- add_context(all).

?- make( global(1, counter 1)).

?- make( vertical(1, pos 1)).
?- make( vertical(1, pos 2)).
?- make( vertical(1, pos 3)).
?- make( vertical(1, pos 4)).
?- make( vertical(1, pos 5)).
?- make( vertical(1, pos 6)).
?- make( vertical(1, pos 7)).
?- make( vertical(1, pos 8)).

conflicts(H1,V1,H2,V2):- H1=H2,!.
conflicts(H1,V1,H2,V2):- V1= V2,!.
conflicts(H1,V1,H2,V2):- abs(H1-H2)=abs(V1-V2).

rule( r1,
      [all],

      global(1, counter H1) and
      vertical(1, pos V1) and
      not_true( 1, queen_position(1, h_pos H2, v_pos V2)
      and evaluate(1, conflicts(H1,V1,H2,V2)))
      -->
      modify(1,1, counter H1+1) and
      remove(2) and
      make(queen_position(1, h_pos H1, v_pos V1))).

end_goal(g1, [all], vertical(-1,pos Y)).
```

This little program solves the ‘8-Queens’ problem. The ‘8-Queens’ problem is a very well known, artificial intelligence problem and has suitable efficient algorithms to solve it. This problem states that eight queens on a chess board can be placed in such a way that no one can beat the other one. Interesting thing here, is that this very short code solves it very efficiently and in a considerably structured manner. The decision taking algorithm, is unique when compared to the other codes implemented in different logic languages. Notice that, there is mainly no strict order in execution. Instead, there are various structures, each are defined separately and work separately.

All the strange appearing concepts will be made clear in the following sections, but we would better, first, state the problem tried to be solved in this particular project.

1.2 Statement Of The Problem

‘ How can an integrated development environment implemented, such that this IDE will enable the user to develop the wanted code, in this rather complicated notation of the SSS expert system shell, without writing any line of code or using just an editor, instead in an very structured way just by clicking buttons and filling in the blanks of the presented windows ?. Moreover, the syntax and type checking of the entered statements are to be done by a specific parser, so the grammar must be developed, also.’

2. DEVELOPMENT ENVIRONMENT OF THE PROJECT

2.1 Why Have We Chosen Borland's Delphi ?

Delphi represents a brand new way to develop applications for Windows. It combines the speed and ease of use of a visual development environment with the power, flexibility, and reusability of a fully object-oriented language, the world's fastest compiler, and leading-edge database technology.

Delphi is a component-based application development environment supporting rapid development of highly efficient Microsoft Windows-based applications with a minimum of coding. Many of the traditional requirements of programming for Windows are handled for the programmer within the Delphi class library, shielding him/her from complicated, or merely repetitive programming tasks.

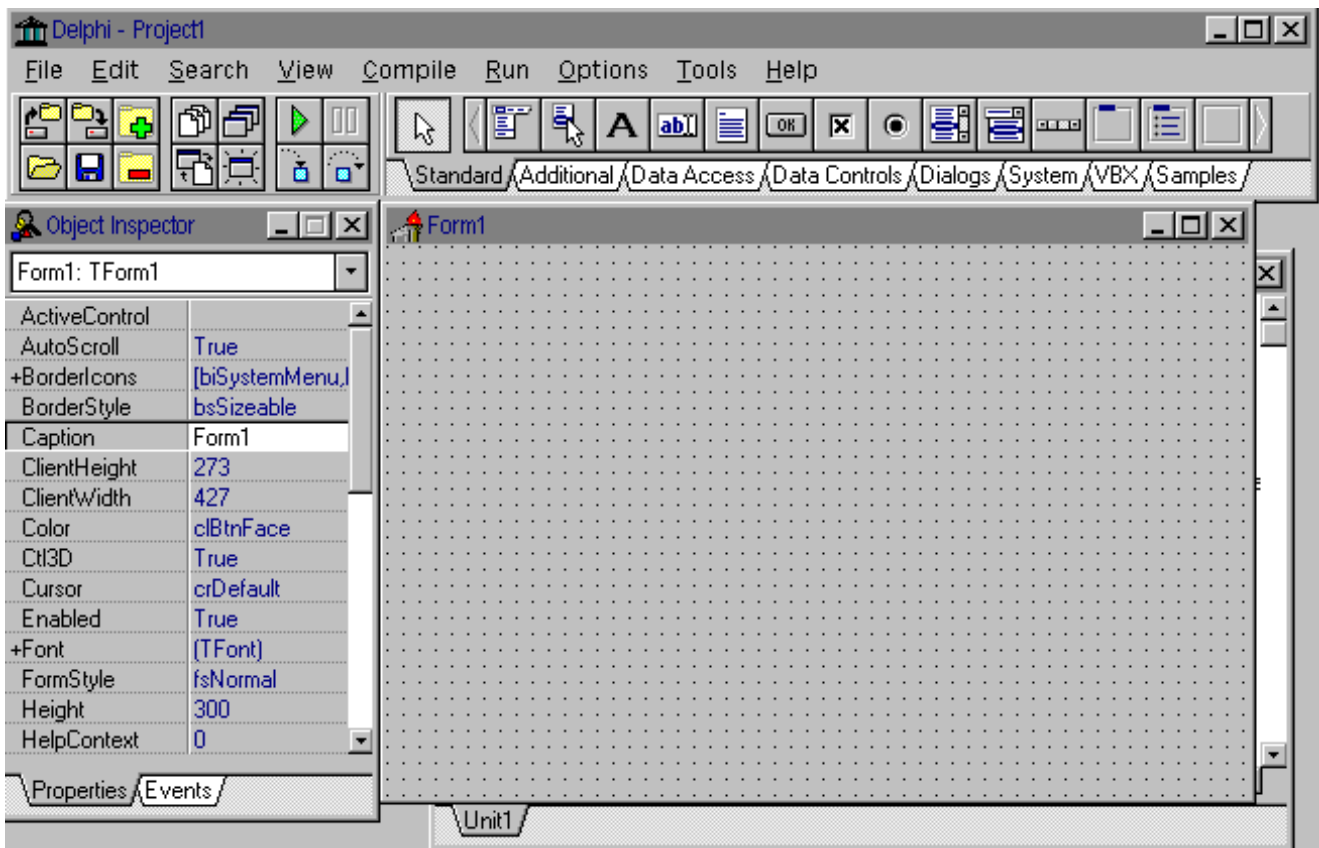
Delphi provides design tools such as application and form templates, so the programmer can quickly create and test his/her application prototype. Then, by using Delphi's rich component set and intuitive code generation, he/she can turn his/her prototypes into robust applications that fit business needs.

Delphi's database tools enable the programmer to develop powerful desktop database and client/ server applications and reports. "Live" data can be viewed at design time, so the programmer knows immediately whether his/her query results are what he/she wants.

These summarize the reasons which made us chose Borland's Delphi as our software development kit.

2.2 Delphi

Issues concerning Delphi, are already explained in the previous part. But what we want to show here about Delphi, is its IDE. It is a very powerful and complicated, integrated development environment that we can here give only a preview :



2.2.1 Forms

Forms are the focal point of nearly every application you develop in Delphi. You use the form like a canvas, placing and arranging components on it to design the parts of

your user interface. Components are the building blocks of Delphi applications. They appear on the Component palette, displayed in the top right-hand part of the screen.

2.2.2 Component Palette

Components are the elements you use to build your Delphi applications. They include all the visible parts of an application, such as dialog boxes and buttons, as well as those that aren't visible while the application is running, such as system timers or Dynamic Data Exchange (DDE) servers.

2.2.3 Object Inspector

The Delphi Object Inspector enables you to easily customize the way a component appears and behaves in your application. The properties and events of the component that is selected in the form are displayed in the Object Inspector. You use the Properties page of the Object Inspector to customize components you've placed on a form (or the form itself), and the Events page to generate and navigate among certain parts of program code, called event handlers. Event handlers are specialized procedures.

2.3 Paradox 5.1 For Windows

This is the latest version of the Borland's Paradox series and also the most sophisticated. We think that, it is better to give little information about this tool, because, it has various capabilities and none of them was essential for this project. In SSS IDE development phase Paradox 5.1 is used only as an ordinary database management system.

The tables are created by this tool and all the other manipulations are done by code under Delphi.

2.4 Windows 95

Windows 95 or version 4.0 is an advanced, 32-bit operating system that runs on 4MB machines and provides excellent response time to both 16 and 32-bit applications. It has the advantage of real multitasking which enables many programs to run at a time. This version has many features beyond the limit of version 3.1. In developing SSS IDE, we have used Windows 95 Version 4.00.950 Release 3.

2.5 Computer System Used

In this project, we used a rather fast 486DX / 50 machine which has a hard disk of 420 MB and a quad speed CD-ROM drive. The amount of main memory, used in developing this project, is 8MB. With this configuration, we have encountered no serious problems in the design phase. We decided to use this total system because of a basic reason. This total system consists of the latest versions of the development kits and a fast computer. As the product team, we thought that it would be easier and more reliable to develop a large project like SSS IDE, in this system.

3. DESIGN OF INTEGRATED DEVELOPMENT ENVIRONMENT SYSTEM

3.1 Introduction To Design

As this project is mainly about developing an integrated development environment for an expert system shell, it would be the best choice to implement it under Windows by using some kind of visual language. 'Delphi' is selected for this purpose and discussion about this selection decision is given in the section concerning 'Delphi'. The underlying expert system shell is also a new development which is known as SSS (three - S). It has some interesting properties as mentioned in the corresponding section of this report. But, there is one more important point concerning the developed interface. This interface is not just an ordinary user interface. The idea was to minimize the amount of code entered by the user and control the structure of the developed code in a strict fashion. With this idea in mind, two separate sections are implemented. The first one is the main 'Code Entrance' section which is discussed and presented in this heading's coverage. The second one is a total 'Parser' which does the job of checking the type consistency and structural correctness of the code generated. The discussion about the parser is given in the next section.

The integrated development environment implemented in this project, has no editor or like construct to manipulate the code directly. Instead of this, the user may click the 'Fact Define' button and fill in the blanks. The actual code is, then, generated by the interface. So, the user never minds about the complexity of nested parenthesis, for example.

3.2 System Forms Of Interaction

In such an environment where the code entrance is tried to be minimized, structuring the interactions with the user is crucial to correct operation of the system. To ensure that structure is consistent, various forms are designed and joined to rather long code parts. There are many different and seemingly strange notations and concepts used in SSS language, all of which is successfully created by the SSS IDE. Following sections will make all these notations and concepts clear. But, we would better to present the SSS main desktop.

3.2.1 SSS Desktop Overview

Desktop is a collection of three main windows. These are 'Main Menu', 'Define Menu' and 'Inventory' windows. Each of these consists of collections of functionality.

3.2.1.1 Main Menu Form

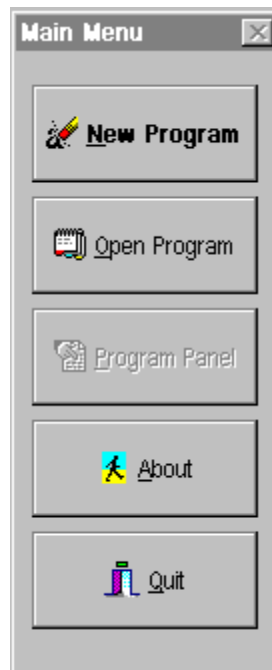


Figure 3-1

'Main Menu' is a collection of various general functionality. With the help of this window, the user can accomplish the main functions presented by the IDE. Upon first initialization of the program, there is no desktop file is opened or created. If the user will do any job useful, he/she has to open a particular work project or create a new, empty project which will be used in the following operations. In the 'Main Menu', 'New Program' option just creates an empty project file and organizes an empty environment to generate the particular code. Clicking 'Open Program' button, opens an existing project and adjusts the environment accordingly. Further user commands enables the user to change the code to be generated. 'Program Panel' takes user to the main place where the actual code to be generated, to be seen and sent to the executor component or the parser component. 'About' is nothing more than an informative screen. 'Quit' is self-explanatory.

3.2.1.1.1 Program Panel Form

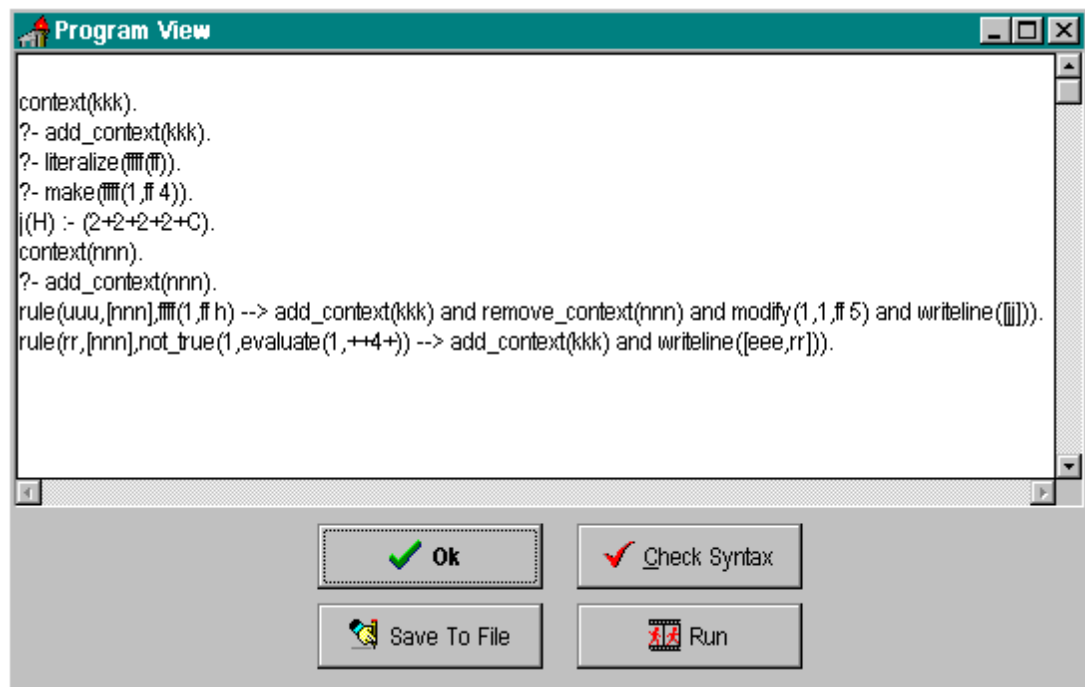


Figure 3-2

'Save To File' operation saves the code to a file so that it may be used further to take a hard copy or for any other purpose. The name of the file generated is the same for the project name. But, to ensure the consistency between the file naming conventions of various operating systems, only the first eight characters of the project name are taken as the file name. The extension is unambiguously determined as '.CLP'. As a note, the limit of the number of characters in the name given to the project and variable identifiers is 32, just like identifier name length restriction in conventional languages.

'Run', passes the code to the executor to execute it. The first operation done, here, is taking a hard copy of the code file into a system file. Then, this file is passed to the executor which is another project that had been implemented in a logical language called CLP(R). This language is very like a dialect of Prolog. The development of the executor is beyond the scope of this project and it is not a part of this IDE project.

'Check File' passes the code generated to the parser to check the syntax consistency. The parser component is developed with the help of a parser generator called 'Bison'. This generator works nearly the same as 'YACC', which is a better known tool, but at no point, stronger than 'Bison'. The discussion about the parser and its generation will be given as a separate section. So, we see little meaning to continue it here.

'OK' button obviously returns the user to the main menu.

3.2.1.1.2 About Form

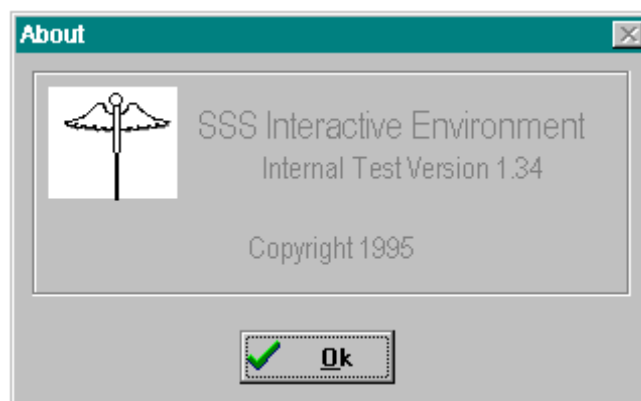


Figure 3-3

Copyright information and the version number of the program is presented on this screen. This is nothing more than a ordinary dialog box, just to inform the user and there is only one button which makes it possible to close it. No other operation can be done during the presentation of this screen to the user.

3.2.1.2 Inventory Form

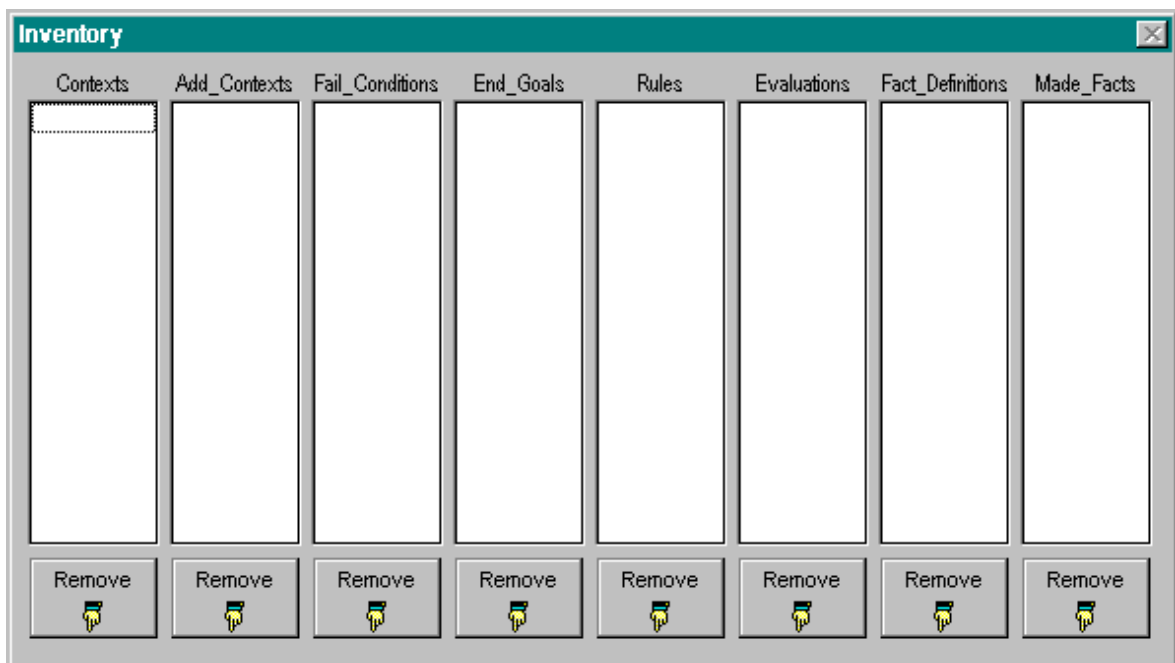


Figure 3-4

Inventory form is one of the main forms that are displayed when the program is first initiated. This is an abstract representation of the current program being developed by the user. Each of the defined structures are shown by their corresponding names, in the corresponding boxes. 'Remove' buttons remove selected statements entered by the user beforehand from the corresponding boxes and also from the program under development. 'Inventory' screen is a necessary tool to show the phases of code development. By referring to this screen, the user can have considerable information about the contents of the program code.

3.2.1.3 Define Menu Form



Figure 3-5

The answer to the question of how the user can add new statements to the current code, is the 'Define Menu'. This menu is the most important part of the IDE, because it implements all that can be done by using an editor. But, the important point is that the operations in this menu, pose a considerable amount of structure and control to the code entrance activity. Also, many unnecessary details concerning the syntax of the notations used, may be passed easily by the user and he/she can concentrate mainly on the concepts he/she wants to implement, nothing else. Moreover, many annoying errors concerning the wrong use of parenthesis, misuse of keywords etc. are easily avoided. Understanding these concepts may be easily attained, if thought one by one. The first operation implemented on define menu is about contexts.

3.2.2 SSS Language Concepts

3.2.2.1 Contexts

Contexts are like common data storage consisting of both data structures and data manipulators. Only the structures and manipulators that are defined in the active context or contexts, are operational. All other structures and manipulators are handled as if they are not defined in anywhere and all of these are not operational, unless the current context or contexts are changed. The collection of contexts defined are usually called as 'Context

Union' and we will use this convention from this point on. Contexts may also be thought as modes of operation. Different operation modes enables different structures and manipulators and disables certain different ones.

There are two operations on contexts. The user may 'Define Context' or 'Make Context'. All the contexts that will be used during the program execution, must be defined beforehand. Defining a context does not enables it, in the first place. Making a context, instead, does it. It means simply to enlarge the current code's context to enable various other structures, fail conditions etc. to become operational. As a note, we must remark that removing a context or context union is also possible.

The selection of 'Define' or 'Make' is done by the following window :

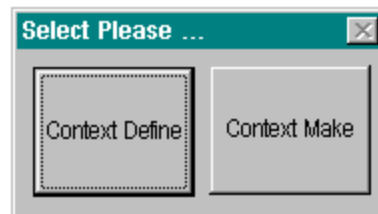


Figure 3-6

The user selects the operation by clicking one of the buttons shown. Regardless of the button clicked, the user presented another window which is the main variable defining window of the program and it is also used at many places of the program for various purposes. We give the layout of this window below for illustrative purposes. But, as this window is used at many more places in the program and also in this report, we will just to refer it in the foregoing points.

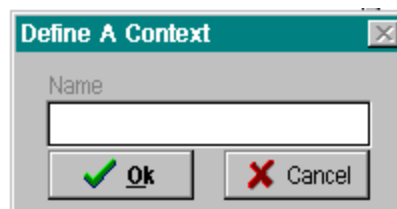


Figure 3-7

The only difference in the various uses of this window is only the difference in the captions. For example, when using to make a context, the caption becomes 'Make A Context' instead of the one you see.

The generated strings, in place of the making and defining contexts are given as follows as examples :

```
context(all).      /* defining a context */  
add_context(all). /* making a context */
```

3.2.2.2 Fail Conditions

'Fail Conditions' show the conditions that will never be true during the program execution in the SSS system. These are used mainly to control the flow of execution and correctness of the decisions created during the execution. Fail conditions like most other structures in the SSS language need context unions defined to work properly. Each fail condition can be only operational under suitable contexts and non operational under others. Basically a fail condition consists of three separable parts. These are the name of the fail condition, context union and facts. The following window is the main window used to enter fail conditions into the system.

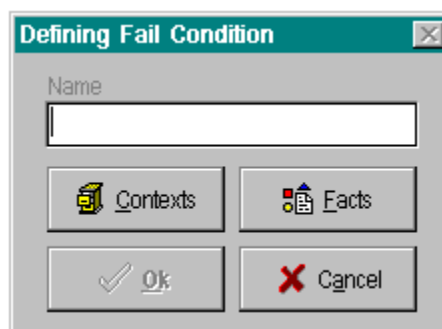


Figure 3-8

Contexts are added to the union defined for the particular fail condition by the help of a special window just designed for the purpose of defining context unions. This window is mainly used in many places during the program development, particularly in the structures that need context unions to operate.

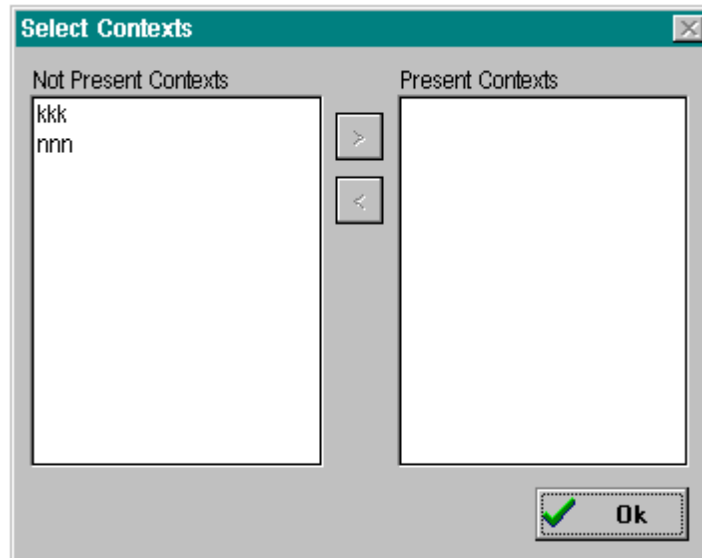


Figure 3-9

The insertion and extraction buttons adjust the union of contexts in which the fail condition will be operational. The left list box contains the contexts defined so far, in the program and which are not in the particular context union. The right list box contains the current context union. As you may intuitively guess, only already defined contexts appear in these list boxes, nothing else.

Fact list of the fail condition is constructed by the help of another special purpose window. This window is also used at many places in the current development phase where any fact list construction is necessary. So, we will give the illustration of the same window only here and not at each place where it is mentioned. Instead, a reference to this window will take place. 'Fact List Define' window is as follows :

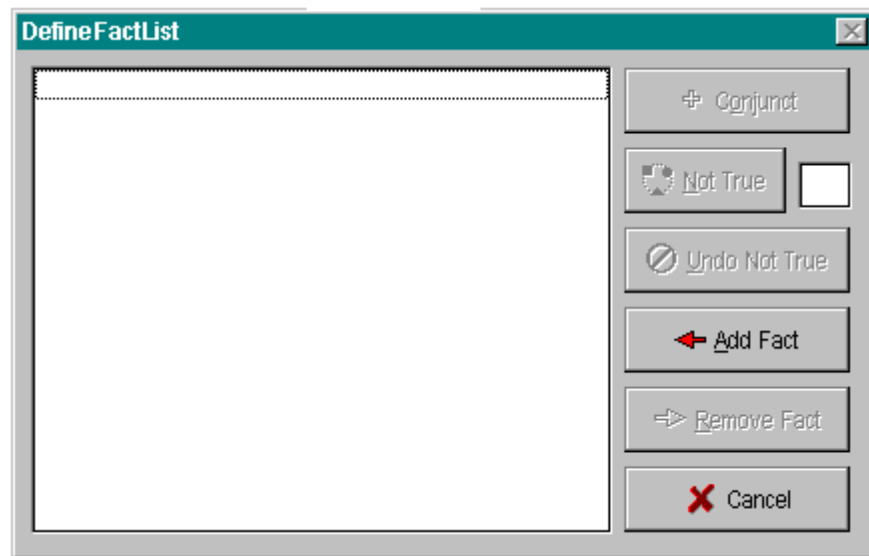


Figure 3-10

Note that the fact list that will be constructed, will show the facts whose total truth value will be always false, during the execution of the code generated. At this point, it would be better to give information about each button on this particular window. As you may noticed, the facts added to the list are collected in the big list box. 'Remove Fact' removes a selected fact from the list box. 'Not True' takes the logical inversion of selected facts. Here the facts in the selected list, are first joined by ands. The little box on the right of the button, defines the probabilistic factor of the not-true fact. This factor is present in nearly all statements of the SSS language and implements the so called 'Fuzzy Logic'. Simply, it determines the probability of using that particular statement in decision making process. A factor of one, means that this fact will be always present in our bag of facts. 'Undo Not True' reverses the effect of the 'Not True' action. 'Conjunct' button joins all the facts in the list box with ands and returns. There is one more operation, here, which needs special consideration : 'Add Fact'. This, simply, adds a new fact to the list box. But, adding a fact is a complex operation and handled by another special window. This window is presented as follows for illustrative purposes :

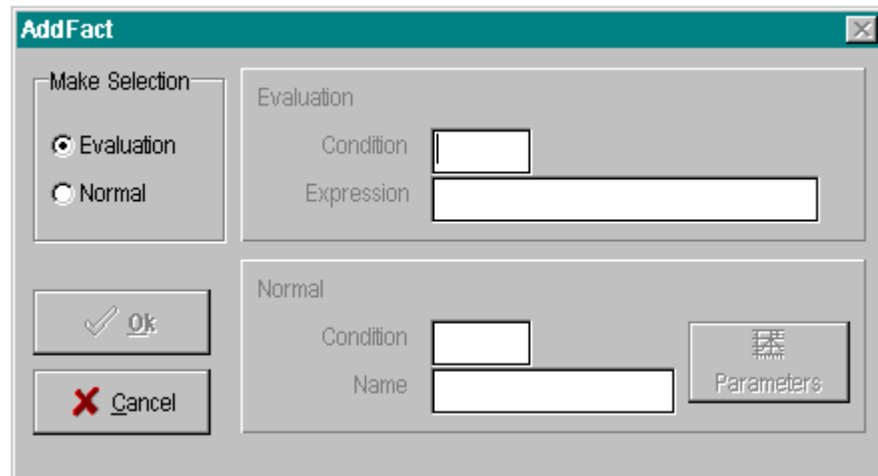


Figure 3-11

Here, a fact may be defined to be a user-defined fact or an evaluation. If it is a normal fact, that is, if it is a user-defined fact, the parameter values of the previously defined fact, must also be specified. Defining a user-defined fact in the first place, is done in the 'Define Menu'. This operation will be described as a separate section in the following lines. Condition fields define the probabilistic factor which indicates the probability of using this fact in the decision making process. Parameter entrance operation also checks whether that fact is predefined or not. According to the condition, an error box is presented to the user.

Some example codes generated by the mentioned operations are given as follows :

```
fail_conditon(f1,[all],evaluate(1,M>0).
```

3.2.2.3 Facts

'Facts' correspond to the logical facts which are present nearly in all logical languages. They implement the logical true assertions in the real world. What they represent is always accepted as having truth value true, directly. During the execution of the program, the facts may be added to the current environment or may be removed from it.

A fact has the same concepts of being defined and being made, just like the contexts. A particular fact has to be defined and enabled to become operational.

Defining a fact, consists of giving a name to it and specify the fields. The field specification is nothing more than collecting them into a list box. Adding a field operation is done by the help of the window in Figure 3-7. Removing is just accomplished by selecting a field and clicking the appropriate button. The window corresponding to the 'Define Fact' operation is given for illustrative purposes as follows :

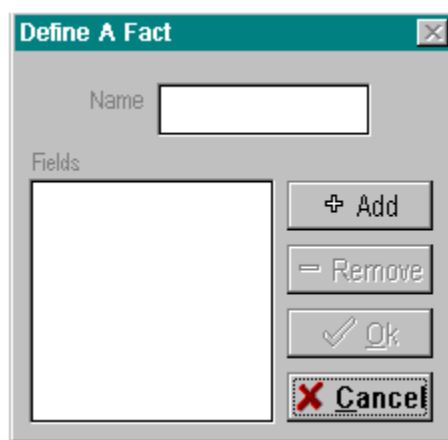


Figure 3-12

Making a fact, means actually using the fact. To use a particular fact, the user defines the fact by using the operation described, in the first place. Then, the user must fill the boxes of probability and name of the corresponding fact's 'Make Fact' window. After giving the actual values for the parameters, the fact is operational and ready to use. The 'Make Fact' window and the example statements created are given below :

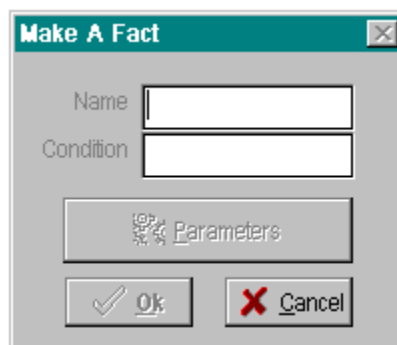


Figure 3-13

As an example of the code generated by these windows, may be given as :

```
/* a fact defined */  
?- literalize( queen_position( h_pos, v_pos)).  
/* a fact made */  
?- make( vertical( 1, pos 8)).
```

3.2.2.4 Evaluations

Evaluations may also be defined as special facts which has formal parameters. All the computation concerning this evaluation, is calculated and the truth value replaces the evaluation statement, where it is used. But, there is an important difference between the normal user-defined facts and evaluations. That is that the evaluations need not to be made. They are operational and ready to use, as soon as they are defined. Definition of the evaluations are done by the help of another special window. This window is given below for illustrative purposes.

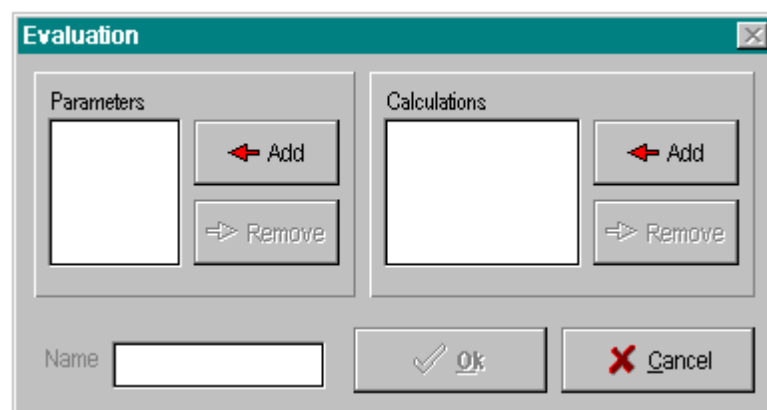


Figure 3-14

To define an evaluation, the user collects the parameters in the left list box and the calculations in the right list box. The generated code statement is given as an example, again. The structural correctness of expressions used and the types of parameters, are

checked by the parser. The example given below may give an idea about the code generated by these operations.

```
conflicts( V1, V2) :- V1 = V2, !.
```

3.2.2.5 Rules

Rules are the most complex structure in the SSS language and these have some special properties to consider particularly. In the most extreme abstraction, rules of SSS corresponds to the rules of other logic languages, like Prolog. Rules are used when you want to say that a fact depends on a group of other facts. In English, we use the word ‘if’ to express a rule.

A rule is a general statement about objects and their relationships. For example, we can say that Fred is a bird, if Fred is an animal and Fred has feathers, and we can also say that Bertram is a bird, if Bertram is an animal and Bertram has feathers. So, we can allow a variable to stand for a different object in each different use of the rule. Within a use of a rule, of course, variables are interpreted consistently as pointed out above.

A rule, consists of a head and a body. The head and the body are connected by the special symbol ‘ \rightarrow ’. The head is named as the left-hand-side and the body as the right-hand-side of the rule. Of course, there are other parts in rule structures, like ‘Context’, ‘Name’ etc. but these are all the same with the other structures.

The idea in the implementation of rules, is to reduce the facts of the left-hand-side into some restricted set of facts in the right-hand-side. The interesting point is that it is nearly always possible in logical problems, to achieve this goal. The left-hand-side is defined as the same as a list of facts defined in the fail condition statement. So, these are generated by the help of the same window in defining fail conditions. This window was presented in Figure 3-10. Contexts are also like the ones defined in fail conditions. The job of defining the corresponding context union is done by using the generic interface of Figure 3-9. But, the right-hand-side is a different concept. So, it is constructed by the help of a new special window interface.

The 'Right-Hand-Side Define' window is presented below for illustrative purposes.

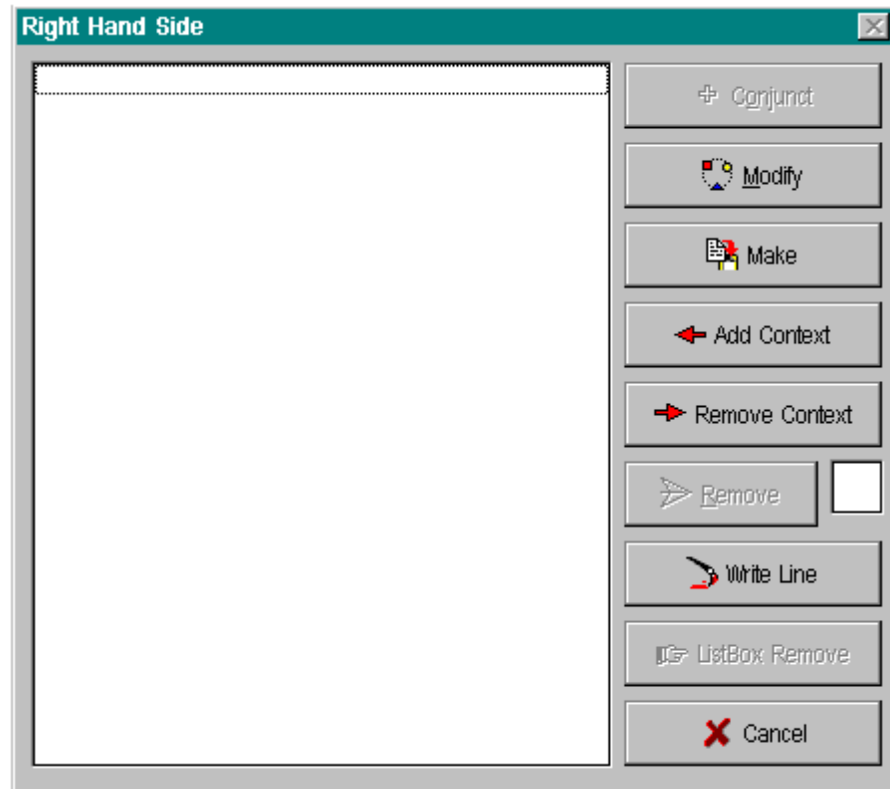


Figure 3-15

The restricted set of facts of right-hand-side consists of 'Modify', 'Make', 'Add Context', 'Remove', 'Remove Context' and 'Write Line'. These are all special facts, mainly used to reduce any set of ordinary facts to a set consisting of themselves. 'Modify' adds a special fact which changes the actual values of a particular fact which resides in the fact set of the left-hand-side. 'Make' adds a special fact which enables some predefined used-fact, so it becomes operational. 'Add Context' adds a special fact that changes the context union of the current code execution by adding a new context. 'Remove Context' adds a special fact that changes the context union of the current code execution by removing a context. 'Remove' adds a special fact that disables a particular fact which resides in the fact set of the left-hand-side and has the offset number given in the little box next to the button. 'Write Line' adds a special fact that prints an informative message on the screen. Each of these is explained in detail, below.

Before explaining the concepts in detail, a note about the other buttons is necessary. 'Conjunct' simply joins all the special facts by ands and returns. 'List Box Remove' removes a right-hand-side fact already entered into the list box.

Clicking the 'Modify' button, presents the following screen to the user :

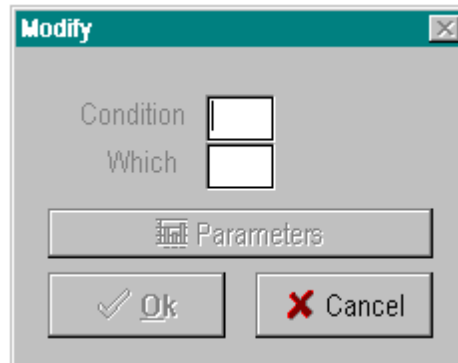


Figure 3-16

Here, the 'condition' determines the probabilistic factor and 'which' shows the offset value of the particular fact to be modified. By the help of the parameters button, new values of the actual parameters are given to the program.

'Make' button shows the 'Make Fact' window which can be remembered as the Figure 3-13. The operations to make a fact are as the same which are described before. 'Add Context' and 'Remove Context' does mainly the same jobs already presented and uses the generic window of Figure 3-7. 'Remove' does not use any window. But 'Write Line' has its associated interface.

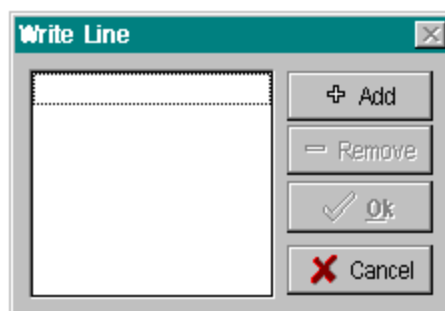


Figure 3-17

The functionality of this interface is nothing more than to collect the items to be printed on the screen. The items are collected into the list box and can be manipulated by 'Add' and 'Remove' buttons.

The statements generated in the case of rules are rather complex ones. An example of this may be given as follows :

```
rule( r1,[all],global(1, counter H1) and
      vertical(1, pos V1) and
      not_true( 1,
      queen_position(1, h_pos H2, v_pos V2)
      and evaluate(1, conflicts(H1,V1,H2,V2)))
      -->
      modify(1,1, counter H1+1) and
      remove(2) and
      make(queen_position(1, h_pos H1, v_pos V1))).
```

3.2.2.6 End Goals

These are mainly the finish conditions indicating the end of the program and are nearly the same as the fail conditions, according to the syntax and code generated. There are nearly no differences between the definition of a fail condition and an end condition. They both use the same window interface, namely Figure 3-8. In the syntax, there is only one difference between these two, which is the use of the keywords : 'end_goal' and 'fail_condition'.

As a note, the end goals, usually, contain only one fact to be satisfied, but this is not a restriction at all.

4. MAIN PARSER COMPONENT OF THE SYSTEM

This section shows how a parser generator, can be used to facilitate the construction of the front end of a compiler, in order to check syntax and type consistency of the generated code. We shall use the LALR parser generator 'Bison' as the basis of our discussion and implementation concepts. It is very similar to the classical parser generator YACC. YACC stands for 'yet another compiler-compiler', reflecting the popularity of the parser generators in the early 1970's, when the first version of YACC was created by S. C. Johnson. YACC is available as a command on the UNIX system and has been used to help implement hundreds of compilers.

4.1 The Parser Generator Bison

A parser can be constructed by using Bison in the manner described as follows. First a file, say 'sssparse.y', containing Bison specification of the parser is prepared. The UNIX system command '\$ bison sssparse.y' transforms the file into a C program called 'sssparse_tab.c' using the classical LALR method. The program 'sssparse_tab' is a representation of an LALR parser written in C, along with other C routines that the user may have prepared. The LALR parsing table is compacted. By compiling 'sssparse_tab.c' along with the 'ly' library that contains the LR parsing program using the command 'cc sssparse_tab.c -ly', we obtain the desired object program that performs the translation specified by the original Bison program. If other procedures are needed, they can be compiled or loaded with 'sssparse_tab.c' just as with any C program.

A Bison source program has three parts :

```
declarations
%%
translation rules
%%
supporting C programs
```

4.1.1 The Declarations Part

There are two optional sections in the declarations part of a Bison program. In the first section, we put ordinary C declarations, delimited by `%{` and `%}`. Here, we place declarations of any temporaries used by the translation rules or procedures of the second and third sections. In many examples, this part contains only the include statement. Also, in the declarations part are declarations of grammar tokens. The statement `%token DIGIT` declares `'DIGIT'` to be a token. Tokens declared in this section, can than be used in the second and third parts of the Bison specification file.

4.1.2 The Translation Rules Part

In the part of the Bison specification after the first `%%` pair, we put the translation rules. Each rule consists of a grammar production and the associated semantic action. A set of productions that we have been writing like

```
<left side> -> <alt 1> | <alt 2> ..
```

would be written in Bison as follows :

```
<left side> : <alt 1> | <alt 2> .. ;
```

In a Bison production, a quoted single character `'c'` is taken to be the terminal symbol `c`, and the unquoted strings of letters and digits not declared to be tokens are taken to be nonterminals. Alternative right sides can be separated by a vertical bar, and a semicolon follows each left side with its alternatives and semantic actions. The first left hand side is taken to be the start symbol.

A Bison semantic action is a sequence of C statements. In a semantic action, the symbol `$$` refers to the attribute value associated with the nonterminal on the left, while

' I ' refers to the value associated with the i th grammar symbol on the right. The semantic action is performed whenever we reduce by the associated production, so normally the semantic action computes a value for '\$\$' in terms of ' I s'. In the Bison specification, we have written the two E-productions

$$E \rightarrow E + T \mid T$$

and their associated semantic actions as

```
expr :    expr '+' term { $$ = $1 + $3; }
      |    term
      ;
```

Note that the nonterminal 'term' in the first production, is the third grammar symbol on the right, while '+' is the second. The semantic action associated with the first production adds the value of the 'expr' and the 'term' on the right and assigns the result as the value for the nonterminal 'expr' on the left. We have omitted the semantic action for the second production altogether, since copying the value is the default action for productions with a single grammar symbol on the right. In general, { \$\$ = \$1; } is the default semantic action.

Notice that, we have added a new starting production

```
line :    expr '\n' { printf("%d\n", $1); }
```

to the Bison specification. This production says that an input to the desk calculator is to be an expression followed by a newline character. The semantic action associated with this production prints the decimal value of the expression followed a newline character.

4.1.3 The Supporting C-routines Part

The third part of a Bison specification consists of supporting C-routines. A lexical analyzer by the name 'yylex()' must be provided. Other procedures such as error recovery routines may be added as necessary.

The lexical analyzer 'yylex()' produces pairs consisting of a token and its associated attribute value. If a token such as 'DIGIT' is returned, the token must be declared in the first section of the Bison specification. The attribute value associated with a token is communicated to the parser through a Bison defined variable 'yylval'.

4.2 SSS Grammar File

As the grammar of the SSS language is given as a part of the parser specification file in the Appendix A, we see little meaning to give it here, again. The interested reader is kindly invited to the Appendix A part to see the SSS language grammar through the full Bison specification of the parser file.

5. A FINAL WORD ABOUT THE PROJECT

SSS language is an interesting experimental effort done in the research area of logic languages and expert system shells. The underlying properties of this recently developed shell, make it sure that the language is really strong, too. This, also, ensures that some future, is present inside the project, if some particular effort may be spent on it. I think, hopefully that, with this particular special project development and the report that you are reading now, considerable amount of the mentioned effort is spent and this process added to its power at some degree.

Design of the integrated development environment and syntax / type checker designed for the SSS language was the main topic of this project and so of this report. This may be right place to mention some difficulties encountered during the development phase of the project.

Designing the windows and organizing them, do not impose any difficulties, for I was rather fluent in the Delphi development environment. The main difficulties was with the generation of the grammar of the language, in the first place. Unfortunately, SSS language is not a very well structured language and some orderless components reside inside its structure. Inserting these structures into an order was a rather difficult operation.

The second problem was about generating huge code concerning special structures like rules, fail conditions, end goals etc. But this problem was an easier one than the first one and could be easily solved.

To summarize, this project was a particularly good topic to think on, design and implement for a student, having an interest in the special programming languages, some topics concerning artificial intelligence, expert system shells and compilers. As a chance, I was interested in all these topics and the project seemed just a suitable one for me and indeed, it was. I am thankful to my instructor, Assistant Professor Mr. Zeki BAYRAM, for preparing such an interesting and joyful project topic for me.

I hope that this report had been a clear and a constructive one for you, as this project was just like that for me. And, I want to thank to you, for all your patience and interest.

APPENDIX A

SOURCE CODE GIVEN TO BISON PARSER GENERATOR

The parser is generated by the following file which is passed to the Bison parser generator. This file has little, where no difference, with the usual YACC source files. Similarities are obvious and the file is self-explanatory to the reader who has the necessary information.

```
%{
#define NULL 0
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

FILE *myfile, *rest;
char tempstr[80],ch;
int howmanysym;

typedef struct tablnode {
                                int tokenfield;
                                char lexemefield[80];
                                } *tablnodeptr,tblnode;

tablnodeptr symboltable=NULL;

%}

%union {
    struct
    {
        char lexe[21];
        int flag;
    } my;
}
%token NUM
```

```

        fprintf(rest, "Ok");
        fclose(rest);
        YYACCEPT;};

statement_list : statement statement_list | ;

context_list : OP_SQ id_list_in CL_SQ ;

id_list_in : expression | id_list_in COMMA expression
            | CONTEXT | id_list_in COMMA CONTEXT;

statement : PROMPT prompt_statement DOT
           | fact_statement DOT
           | ev_statement DOT ;

prompt_statement : LITERALIZE OP_PARAN ID lit_arg_list CL_PARAN |
                  ADD_CONTEXT OP_PARAN ID CL_PARAN |
                  MAKE OP_PARAN ID make_arg_list CL_PARAN |
                  REMOVE_CONTEXT OP_PARAN ID CL_PARAN |
                  REMOVE OP_PARAN texp CL_PARAN |
                  MODIFY OP_PARAN texp COMMA texp COMMA make_id_in
                  CL_PARAN |
                  WRITELINE OP_PARAN OP_SQ id_list_in CL_SQ
                  CL_PARAN ;

lit_arg_list : OP_PARAN id_list_in CL_PARAN ;

make_arg_list : OP_PARAN make_id_list CL_PARAN ;

make_id_list : texp COMMA make_id_in | texp ;

make_id_in : ID texp | make_id_in COMMA ID texp ;

fact_statement : CONTEXT OP_PARAN ID CL_PARAN |
                FAIL_CONDITION OP_PARAN ID COMMA context_list
                COMMA fact_list CL_PARAN |
                END_GOAL OP_PARAN ID COMMA context_list COMMA
                fact_list CL_PARAN |
                RULE OP_PARAN ID COMMA context_list COMMA
                fact_list ARROW rule_right CL_PARAN ;

rule_right : prompt_statement | prompt_statement AND rule_right ;

ev_statement : ev_id_part IMPLIES expression_list |

```

```
        ev_id_part IMPLIES expression |
        ev_id_part ;

ev_id_part : ID OP_PARAN id_list_in CL_PARAN ;

expression_list : expression_list COMMA expression |
                expression COMMA expression |
                OP_PARAN expression_list CL_PARAN ;

expression : texp EQ texp
            | texp LT texp | texp GT texp | texp LE texp |
            texp GE texp | texp | EXCLAM ;

texp : texp ADD_OP sexp |
      texp SUB_OP sexp |
      sexp |
      ABS OP_PARAN texp CL_PARAN ;

sexp : sexp MUL_OP zexp |
      sexp DIV_OP zexp |
      zexp ;

zexp : NUM | VAR | ID
      | OP_PARAN expression CL_PARAN
      | SUB_OP zexp
      | ev_id_part ;

fact_list : onefact AND fact_list | onefact ;

onefact : ID OP_PARAN one_in CL_PARAN |
         NOT_TRUE OP_PARAN texp COMMA fact_list CL_PARAN |
         EVALUATE OP_PARAN texp COMMA expression_list CL_PARAN |
         EVALUATE OP_PARAN texp COMMA expression CL_PARAN ;

one_in : texp one_in_in ;

one_in_in : COMMA fact_in_list | ;

fact_in_list : ID texp | fact_in_list COMMA ID texp ;
%%

void freetable(void)
```

```
{
    free(symboltable);
}

int install_ID(char* tempstr,int mytoken)
{
    int i;
    for(i=0;i<howmanysym;i++)
        if (strcmp(symboltable[i].lexemefield,tempstr)==0)
            return symboltable[i].tokenfield;
    strcpy(symboltable[howmanysym].lexemefield,tempstr);
    symboltable[howmanysym].tokenfield=mytoken;
    howmanysym=howmanysym+1;
    return ID;
}

int install_VAR(char* tempstr)
{
    int i;
    for(i=0;i<howmanysym;i++)
        if (strcmp(symboltable[i].lexemefield,tempstr)==0)
            return symboltable[i].tokenfield;
    strcpy(symboltable[howmanysym].lexemefield,tempstr);
    symboltable[howmanysym].tokenfield=VAR;
    howmanysym=howmanysym+1;
    return VAR;
}

int fail(int state)
{
    ungetc(ch,myfile);
    strcpy(tempstr,"");
    switch(state)
    {
        case 0 : state=3;break;
        case 3 : state=6;break;
        case 6 : state=11;break;
        case 11 : state=15;break;
        case 15 : state=18;break;
        case 18 : state=21;break;
    }
}
```

```
    case 21 : state=24;break;
    case 24 : state=26;break;
    case 26 : state=32;break;
    case 32 : state=35;break;
    case 35 : state=41;break;
    case 41 : state=49;break;
    default : printf("\n Lexical Error ..");
             exit(1);
}
return state;
}
```

```
int install_NUM(char* tempstr)
{
    int i;
    for(i=0;i<howmanysym;i++)
        if (strcmp(symboltable[i].lexemefield,tempstr)==0)
            return symboltable[i].tokenfield;
    strcpy(symboltable[howmanysym].lexemefield,tempstr);
    symboltable[howmanysym].tokenfield=NUM;
    howmanysym=howmanysym+1;
    return NUM;
}
```

```
int fileopen(char *filename)
{
    myfile=fopen(filename,"r");
    if (myfile==NULL)
    {
        printf("Can't Open File..\n");
        exit(1);
    }
    printf("Opened %s ... \n",filename);
    return 0;
}
```

```
int yylex()
```

```
{
    char *achar="";
    int state=0;
    strcpy(tempstr,"");
    while (!(feof(myfile)))
    {
        strcpy(achar,"");
        switch(state)
        {
            case 0 : ch=getc(myfile);
                    if (islower(ch))
                    {
                        state=1;
                        *achar=ch;
                        strcat(tempstr,achar);
                    }
                    else state=fail(state);
                    break;

            case 1 : ch=getc(myfile);
                    if (isalpha(ch) || isdigit(ch) ||
                    (ch=='_'))
                    {
                        state=1;
                        *achar=ch;
                        strcat(tempstr,achar);
                    }
                    else state=2;
                    break;

            case 2 : ungetc(ch,myfile);
                    return install_ID(tempstr,ID);

            case 3 : ch=getc(myfile);
                    if (isupper(ch) || (ch=='_'))
                    {
                        state=4;
                        *achar=ch;
                        strcat(tempstr,achar);
                    }
                    else state=fail(state);
                    break;
        }
    }
}
```

```
case 4 : ch=getc(myfile);
        if (isalpha(ch) || isdigit(ch) ||
            (ch=='_'))
        {
            state=4;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else state=5;
        break;

case 5 : ungetc(ch,myfile);
        return install_VAR(tempstr);

case 6 : ch=getc(myfile);
        if (ch=='-') state=7;
            else state=fail(state);
        break;

case 7 : ch=getc(myfile);
        if (ch=='-') state=8;
            else state=10;
        break;

case 8 : ch=getc(myfile);
        if (ch=='>') state=9;
            else state=fail(state);
        break;

case 9 : return ARROW;

case 10 : ungetc(ch,myfile);
        return SUB_OP;

case 11 : ch=getc(myfile);
        if (ch=='+') state=12;
            else if (ch=='*') state=13;
                else if (ch=='/') state=14;
                    else state=fail(state);
        break;

case 12 : return ADD_OP;
```

```
case 13 : return MUL_OP;

case 14 : ch=getc(myfile);
        if (ch=='*') state=37;
        else state=36;
        break;

case 15 : ch=getc(myfile);
        if (ch=='(') state=16;
        else if (ch==')') state=17;
        else state=fail(state);
        break;

case 16 : return OP_PARAN;

case 17 : return CL_PARAN;

case 18 : ch=getc(myfile);
        if (ch=='[') state=19;
        else if (ch==']') state=20;
        else state=fail(state);
        break;

case 19 : return OP_SQ;

case 20 : return CL_SQ;

case 21 : ch=getc(myfile);
        if (ch==':') state=22;
        else state=fail(state);
        break;

case 22 : ch=getc(myfile);
        if (ch=='-') state=23;
        else state=fail(state);
        break;

case 23 : return IMPLIES;

case 24 : ch=getc(myfile);
        if (ch=='.') state=25;
        else state=fail(state);
        break;
```

```
case 25 : return DOT;

case 26 : ch=getc(myfile);
        if (isdigit(ch))
        {
            state=27;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else state=fail(state);
        break;

case 27 : ch=getc(myfile);
        if (isdigit(ch))
        {
            state=27;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else if (ch=='.')
        {
            state=29;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else state=28;
        break;

case 28 : ungetc(ch,myfile);
        return install_NUM(tempstr);

case 29 : ch=getc(myfile);
        if (isdigit(ch))
        {
            state=30;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else state=fail(state);
        break;

case 30 : ch=getc(myfile);
```

```
        if (isdigit(ch))
        {
            state=30;
            *achar=ch;
            strcat(tempstr,achar);
        }
        else state=31;
        break;

case 31 : ungetc(ch,myfile);
        return install_NUM(tempstr);

case 32 : ch=getc(myfile);
        if (ch=='?') state=33;
            else state=fail(state);
        break;

case 33 : ch=getc(myfile);
        if (ch=='-') state=34;
            else state=fail(state);
        break;

case 34 : return PROMPT;

case 35 : ch=getc(myfile);
        if ((ch==' ') || (ch=='\n') ||
            (ch=='\t'))
            state=40;
            else state=fail(state);
        break;

case 36 : ungetc(ch,myfile);
        return DIV_OP;

case 37 : ch=getc(myfile);
        if (ch=='*') state=38;
            else state=37;
        break;

case 38 : ch=getc(myfile);
        if (ch=='*') state=38;
            else if (ch=='/') state=39;
                else state=37;
```

```
        break;

case 39 : state=0;
        break;

case 40 : ch=getc(myfile);
        if ((ch==' ') || (ch=='\n') ||
            (ch=='\t'))
            state=40;
        else {
            ungetc(ch,myfile);
            state=0;
        }
        break;

case 41 : ch=getc(myfile);
        if (ch=='<') state=42;
            else if (ch=='>') state=43;
            else if (ch=='=') state=44;
            else state=fail(state);
        break;

case 42 : ch=getc(myfile);
        if (ch=='=') state=45;
            else state=46;
        break;

case 43 : ch=getc(myfile);
        if (ch=='=') state=47;
            else state=48;
        break;

case 44 : return EQ;

case 45 : return LE;

case 46 : ungetc(ch,myfile);
        return LT;

case 47 : return GE;

case 48 : ungetc(ch,myfile);
        return GT;
```

```
        case 49 : ch=getc(myfile);
                if (ch==',') state=50;
                    else if (ch=='!') state=51;
                        else state=fail(state);
                break;

        case 50 : return COMMA;

        case 51 : return EXCLAM;

    }
}
return PROB;
}

int inittable()
{
    symboltable=(tblenodeptr)calloc(400,sizeof(tblnode));
    install_ID("and",AND);
    install_ID("literalize",LITERALIZE);
    install_ID("context",CONTEXT);
    install_ID("add_context",ADD_CONTEXT);
    install_ID("make",MAKE);
    install_ID("abs",ABS);
    install_ID("modify",MODIFY);
    install_ID("remove",REMOVE);
    install_ID("end_goal",END_GOAL);
    install_ID("fail_condition",FAIL_CONDITION);
    install_ID("evaluate",EVALUATE);
    install_ID("rule",RULE);
    install_ID("remove_context",REMOVE_CONTEXT);
    install_ID("writeline",WRITELINE);
    install_ID("not_true",NOT_TRUE);
    return 0;
}

int main(int argc, char *argv[])
{
    rest=fopen("result.sss","w");
    fprintf(rest,"Er");
    fclose(rest);
}
```

```
clrscr();
fileopen(argv[1]);
inittable();
yyvsparse();
fclose(myfile);
exit(0);
}

yyerror(char *s)
{
printf("%s\n",s);
}
```

APPENDIX B

SOURCE CODES OF DELPHI

This appendix is devoted to the actual source codes of the project. Note that this project was implemented in Delphi and Delphi organizes the codes in separate units. So, the code of the project is given in terms of units.

```
{-----}

unit Sss_u1;

interface

uses

  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sss_u5, Sss_u7, DB, DBTables, Sss_u16,
  Sss_u18, Sss_u20, Sss_u21;

type

  TDefineMenu = class(TForm)
```

```
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    Mem1: TMemo;
    Table1: TTable;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    DefineMenu: TDefineMenu;

implementation

{$R *.DFM}

procedure TDefineMenu.BitBtn1Click(Sender: TObject);
var temp:string;
begin
    SelectC.Hint := Hint;
    SelectC.ShowModal;
end;

procedure TDefineMenu.BitBtn2Click(Sender: TObject);
begin
    DefineFail.Caption := 'Defining Fail Condition';
    DefineFail.Hint := Hint;
    DefineFail.ShowModal;
    Mem1.Lines.Clear;
    if DefineFail.FailCond <> '' then
    begin
        DefineFail.FailCond := 'fail_condition' + DefineFail.FailCond;
        Mem1.Lines.Add(DefineFail.FailCond);
        Table1.Append;
        Table1.FieldName('Type').AsString := 'fail_condition';
        Table1.FieldName('Name').AsString := DefineFail.Edit1.Text;
```

```
        Table1.FieldName('String').Assign(Mem01.Lines);
        Table1.FieldName('Project').AsString := Hint;
        Table1.Post;
    end
end;

procedure TDefineMenu.BitBtn3Click(Sender: TObject);
begin
    DefineFail.Hint := Hint;
    DefineFail.Caption := 'Defining End Goal';
    DefineFail.ShowModal;
    if DefineFail.FailCond <> '' then
    begin
        Mem01.Lines.Clear;
        DefineFail.FailCond := 'end_goal' + DefineFail.FailCond;
        Mem01.Lines.Add(DefineFail.FailCond);
        Table1.Append;
        Table1.FieldName('Type').AsString := 'end_goal';
        Table1.FieldName('Name').AsString := DefineFail.Edit1.Text;
        Table1.FieldName('String').Assign(Mem01.Lines);
        Table1.FieldName('Project').AsString := Hint;
        Table1.Post;
    end
end;

procedure TDefineMenu.BitBtn6Click(Sender: TObject);
begin
    SelectF.Hint := Hint;
    SelectF.ShowModal;
end;

procedure TDefineMenu.BitBtn5Click(Sender: TObject);
begin
    Eval.Hint := Hint;
    Eval.ShowModal;
end;

procedure TDefineMenu.BitBtn4Click(Sender: TObject);
begin
    DefineRule.Hint := Hint;
    DefineRule.ShowModal;
    if DefineRule.RuleList <> '' then
    begin
        Mem01.Lines.Clear;
        Mem01.Lines.Add(DefineRule.RuleList);
        Table1.Append;
        Table1.FieldName('Type').AsString := 'rule';
```

```
Table1.FieldName('Name').AsString := DefineRule.Edit1.Text;
Table1.FieldName('String').Assign(Mem01.Lines);
Table1.FieldName('Project').AsString := Hint;
Table1.Post;
end
end;

end.

{-----}

unit Sss_u2;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sss_u1, Sss_u3, Sss_u14, Sss_u19, DB,
  DBTables, Sss_u5, Sss_u6, Sss_u10, Sss_u26, ToolHelp;

type
  TMainMenu = class(TForm)
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    Query1: TQuery;
    Query2: TQuery;
    procedure BitBtn1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainMenu: TMainMenu;

implementation
```

```
{ $R *.DFM }

procedure TMainMenu.BitBtn1Click(Sender: TObject);
begin
    DefineMenu.Close;
    Inside.Close;
    Copyright.Close;
    Close;
    Halt;
end;

procedure TMainMenu.FormShow(Sender: TObject);
begin
    DefineMenu.Show;
    Inside.Show;
    Copyright.Show;
end;

procedure TMainMenu.BitBtn2Click(Sender: TObject);
begin
    Viewer.Hint := Hint;
    Viewer.Show
end;

procedure TMainMenu.BitBtn3Click(Sender: TObject);
begin
    AboutBox.ShowModal;
end;

procedure TMainMenu.BitBtn5Click(Sender: TObject);
begin
    DefineVariable.Caption := 'Enter File Name';
    DefineVariable.ShowModal;
    if DefineVariable.Variable <> '' then
    begin
        Query1.Close;
        Query1.Params[0].AsString := DefineVariable.Variable;
        Query1.Open;
        if not(Query1.EOF) then
        begin
            ErrorBox.Label1.Caption := 'This file is already present..';
            ErrorBox.ShowModal;
        end else
        begin
            DefineMenu.Hint := DefineVariable.Variable;
            Inside.Hint := DefineVariable.Variable;
            Hint := DefineVariable.Variable;
        end
    end;
end;
```

```
    BitBtn2.Enabled := True;
    DefineMenu.BitBtn1.Enabled := True;
    DefineMenu.BitBtn2.Enabled := True;
    DefineMenu.BitBtn3.Enabled := True;
    DefineMenu.BitBtn4.Enabled := True;
    DefineMenu.BitBtn5.Enabled := True;
    DefineMenu.BitBtn6.Enabled := True;
  end
end
end;

procedure TMainMenu.BitBtn6Click(Sender: TObject);
begin
  Query2.Close;
  Query2.Open;
  if Query2.EOF then
  begin
    ErrorBox.Label1.Caption := 'No File Present ..';
    ErrorBox.ShowModal;
  end else
  begin
    RemoveFact.ListBox1.Items.Clear;
    while not(Query2.EOF) do
    begin
      RemoveFact.ListBox1.Items.Add(Query2.Fields[0].AsString);
      Query2.Next;
    end;
    RemoveFact.Caption := 'Enter File Name';
    RemoveFact.ShowModal;
    if RemoveFact.Removed<> '' then
    begin
      DefineMenu.Hint := RemoveFact.Removed;
      Inside.Hint := RemoveFact.Removed;
      Hint := RemoveFact.Removed;
      BitBtn2.Enabled := True;
      DefineMenu.BitBtn1.Enabled := True;
      DefineMenu.BitBtn2.Enabled := True;
      DefineMenu.BitBtn3.Enabled := True;
      DefineMenu.BitBtn4.Enabled := True;
      DefineMenu.BitBtn5.Enabled := True;
      DefineMenu.BitBtn6.Enabled := True;
    end
  end
end;

end.
```

```
{-----}

unit Sss_u3;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, DB, DBTables, ShellAPI, Sss_u6,
  ExtCtrls;

type
  TViewer = class(TForm)
    Mem1: TMemo;
    BitBtn1: TBitBtn;
    Table1: TTable;
    ListBox1: TListBox;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Timer1: TTimer;
    procedure FormResize(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Viewer: TViewer;

implementation

{$R *.DFM}

procedure TViewer.FormResize(Sender: TObject);
begin
  Mem1.Left := 1;
  Mem1.Top := 1;
  Mem1.Width := Viewer.Width - 10;
end;
```

```
Memol.Height := Viewer.Height - 120;
BitBtn1.Left := (Viewer.Width DIV 2) - 120;
BitBtn2.Left := (Viewer.Width DIV 2) - 120;
BitBtn3.Left := (Viewer.Width DIV 2) + 10;
BitBtn4.Left := (Viewer.Width DIV 2) + 10;
BitBtn1.Top := Viewer.Height - 110;
BitBtn2.Top := Viewer.Height - 70;
BitBtn3.Top := Viewer.Height - 110;
BitBtn4.Top := Viewer.Height - 70;
end;

procedure TViewer.BitBtn1Click(Sender: TObject);
begin
  Close
end;

procedure TViewer.FormShow(Sender: TObject);
var i : integer;
begin
  Memol.Lines.Clear;
  Memol.Lines.Add(Chr(13));
  Table1.First;
  while not(Table1.EOF) do
  begin
    if Table1.FieldName('Project').AsString = Hint then
    begin
      ListBox1.Items.Assign(Table1.FieldName('String'));
      for i:=0 to ListBox1.Items.Count-1 do
        Memol.Lines.Add(ListBox1.Items[i]);
      end;
      Table1.Next
    end
  end
end;

procedure TViewer.BitBtn2Click(Sender: TObject);
var i : integer;
    temp : string;
begin
  temp := '';
  for i:=1 to length(Hint) do
    if i<=8 then temp := temp + Hint[i];
  temp := temp + '.clp';
  Memol.Lines.SaveToFile(temp);
end;

procedure TViewer.BitBtn3Click(Sender: TObject);
var i : integer;
```

```
        temp : string;
        t2 : array [0..79] of char;
begin
    temp := '';
    for i:=1 to length(Hint) do
        if i<=8 then temp := temp + Hint[i];
    temp := temp + '.clp';
    Memol.Lines.SaveToFile(temp);
    StrPCopy(t2,temp);
    ShellExecute(0,NIL,'sssparse.exe',t2,'',SW_SHOW);
    Timer1.Enabled := True;
end;

procedure TViewer.BitBtn4Click(Sender: TObject);
begin
    ErrorBox.Label1.Caption := 'Not Available In This Version ..';
    ErrorBox.ShowModal;
end;

procedure TViewer.Timer1Timer(Sender: TObject);
var myfile : TextFile;
    temp : string;
begin
    Timer1.Enabled := False;
    AssignFile(myfile,'result.sss');
    Reset(myfile);
    Read(myfile,temp);
    CloseFile(myfile);
    if temp = 'Ok'
        then ErrorBox.Label1.Caption := 'Syntax Check Ok ..'
        else ErrorBox.Label1.Caption := 'Error In Syntax ..';
    ErrorBox.Caption := 'Parser Result';
    ErrorBox.ShowModal;
    ErrorBox.Caption := 'Error Detected ...';
end;

end.

{-----}

unit Sss_u4;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
```

```
Forms, Dialogs, DB, DBTables, StdCtrls, Buttons;

type
  TForm4 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    BitBtn1: TBitBtn;
    Table1: TTable;
    Memo1: TMemo;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.DFM}

procedure TForm4.BitBtn1Click(Sender: TObject);
var temp : String;
begin
  if Edit1.Text[0] in ['a'..'z','_'] then
  begin
    Table1.Append;
    Table1.Fields[0].AsString:='context';
    temp := 'context(' + Edit1.Text + ').';
    Memo1.Lines.Add(temp);
    Table1.Fields[1].Assign(Memo1.Lines);
    Table1.Post;
    Form4.Close
  end
end;

end.

{-----}

unit Sss_u5;

interface
```

```
uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons, DB, DBTables, Sss_u6;

type
    TDefineVariable = class(TForm)
        Edit1: TEdit;
        BitBtn1: TBitBtn;
        Label1: TLabel;
        BitBtn2: TBitBtn;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure Edit1Change(Sender: TObject);
        procedure FormShow(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Variable : string;
    end;

var
    DefineVariable: TDefineVariable;

implementation

{$R *.DFM}

procedure TDefineVariable.BitBtn1Click(Sender: TObject);
begin
    if Edit1.Text[1] in ['a'..'z', '_']
    then begin
        Variable := Edit1.Text;
        Close;
    end
    else begin
        ErrorBox.Label1.Caption := 'First Letter Must Be In "a".."z", "_";
        ErrorBox.ShowModal;
    end
end;

procedure TDefineVariable.BitBtn2Click(Sender: TObject);
begin
    Variable := '';
    Close;
end;
```

```
procedure TDefineVariable.Edit1Change(Sender: TObject);
begin
  if Edit1.Text = ''
  then BitBtn1.Enabled := False
  else BitBtn1.Enabled := True;
end;

procedure TDefineVariable.FormShow(Sender: TObject);
begin
  Edit1.Text := '';
end;

end.

{-----}

unit Sss_u6;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TErrorBox = class(TForm)
    Panell: TPanel;
    OKButton: TBitBtn;
    ProgramIcon: TImage;
    Label1: TLabel;
    procedure OKButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ErrorBox: TErrorBox;

implementation

{$R *.DFM}

procedure TErrorBox.OKButtonClick(Sender: TObject);
begin
```

```
        Close
    end;

end.

{-----}

unit Sss_u7;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons, Sss_u12, Sss_u9, ExtCtrls, DB,
DBTables;

type
    TDefineFail = class(TForm)
        Label1: TLabel;
        Edit1: TEdit;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        BitBtn4: TBitBtn;
        Timer1: TTimer;
        Memo1: TMemo;
        Table1: TTable;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn4Click(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure FormShow(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure Edit1Change(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        FailCond : string;
    end;

var
    DefineFail: TDefineFail;
    ConArr : Array [1..25] of string;
    ContextNum : integer;
    Facted,Named : boolean;
```

```
implementation

{$R *.DFM}

procedure TDefineFail.BitBtn1Click(Sender: TObject);
var i : integer;
begin
    ContextAdd.Hint := Hint;
    ContextAdd.SrcList.Items.Clear;
    ContextAdd.DstList.Items.Clear;
    for i:=1 to ContextNum do
        ContextAdd.DstList.Items.Add(ConArr[i]);
    ContextAdd.ShowModal;
    if ContextAdd.Ok then
        begin
            ContextNum := 0;
            for i:=0 to ContextAdd.DstList.Items.Count-1 do
                begin
                    ContextNum := ContextNum + 1;
                    ConArr[ContextNum] := ContextAdd.DstList.Items[i]
                end
            end
        end;
end;

procedure TDefineFail.BitBtn4Click(Sender: TObject);
begin
    FailCond := '';
    Close;
end;

procedure TDefineFail.BitBtn3Click(Sender: TObject);
var tempstr : string;
    i : integer;
begin
    tempstr := ConArr[1];
    for i:=2 to ContextNum do
        tempstr := tempstr + ',' + ConArr[i];
    tempstr := '(' + Edit1.Text + ',[' + tempstr + '], ' +
        DefineFactList.FactList + ')';
    FailCond := tempstr;
    Close;
end;

procedure TDefineFail.Timer1Timer(Sender: TObject);
begin
    if (ContextNum = 0) or not(Facted) or not(Named)
```

```
    then BitBtn3.Enabled := False
    else BitBtn3.Enabled := True;
end;

procedure TDefineFail.FormShow(Sender: TObject);
begin
    Edit1.Text := '';
    ContextNum := 0;
    Facted := false;
    Named := false;
end;

procedure TDefineFail.BitBtn2Click(Sender: TObject);
begin
    DefineFactList.Hint := Hint;
    DefineFactList.ShowModal;
    if DefineFactList.FactList <> '' then Facted := True;
end;

procedure TDefineFail.Edit1Change(Sender: TObject);
begin
    if (Edit1.Text = '') or (Edit1.Text[1] in ['0'..'9'])
        then Named := False
        else Named := True;
end;

end.

{-----}

unit Sss_u8;

interface

uses

    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons;

type

    TForm7 = class(TForm)
        BitBtn4: TBitBtn;
        Label1: TLabel;
        Edit1: TEdit;
        procedure BitBtn4Click(Sender: TObject);
    private
        { Private declarations }
    public
```

```
        { Public declarations }
    end;

var
    Form7: TForm7;

implementation

{$R *.DFM}

procedure TForm7.BitBtn4Click(Sender: TObject);
begin
    if Edit1.Text[1] in ['a'..'z','_']
    then Hint := Edit1.Text;
    Form7.Hide;
end;

end.
{-----}

unit Sss_u9;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons, ExtCtrls,Sss_u10,Sss_u11;

type
    TDefineFactList = class(TForm)
        ListBox1: TListBox;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        BitBtn4: TBitBtn;
        Timer1: TTimer;
        BitBtn5: TBitBtn;
        BitBtn6: TBitBtn;
        ListBox2: TListBox;
        Edit1: TEdit;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn4Click(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure BitBtn5Click(Sender: TObject);
        procedure FormShow(Sender: TObject);
        procedure BitBtn6Click(Sender: TObject);
    end;
end.
```

```
        procedure BitBtn3Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        FactList : string;
    end;

var
    DefineFactList: TDefineFactList;
    FactArr,FactName : array [1..30] of string;
    FactNum : integer;

implementation

{$R *.DFM}

procedure TDefineFactList.BitBtn1Click(Sender: TObject);
var temp:string;
    i : integer;
begin
    temp := ListBox1.Items[0];
    for i:=1 to ListBox1.Items.Count-1 do
        temp := temp + ' and ' + ListBox1.Items[i];
    FactList := temp;
    Close;
end;

procedure TDefineFactList.BitBtn4Click(Sender: TObject);
begin
    FactList := '';
    Close;
end;

procedure TDefineFactList.Timer1Timer(Sender: TObject);
var i,j,tt,ee:integer;
    Btn6:boolean;
begin
    if ListBox1.Items.Count = 0
    then begin
        BitBtn1.Enabled := False;
        BitBtn2.Enabled := False;
        BitBtn5.Enabled := False;
    end
    else begin
        BitBtn1.Enabled := True;
        if Edit1.Text <> '' then BitBtn2.Enabled := True
```

```
                else BitBtn2.Enabled := False;
            BitBtn5.Enabled := True;
        end;
    Btn6 := false;
    for i:=0 to ListBox1.Items.Count-1 do
        for j:=1 to FactNum do
            if ListBox1.Items[i] = FactArr[j] then Btn6:=true;
        if Btn6 then BitBtn6.Enabled := True
            else BitBtn6.Enabled := False;
        end;
    end;

    procedure TDefineFactList.BitBtn2Click(Sender: TObject);
    var i:integer;
        nottrue,result : string;
        delarr : array [1..20] of integer;
        dely : integer;
    begin
        dely := 0;
        if ListBox1.SelCount > 0 then
            begin
                nottrue := '';
                for i:=0 to ListBox1.Items.Count-1 do
                    if ListBox1.Selected[i] then
                        begin
                            nottrue:=ListBox1.Items[i];
                            dely := dely + 1;
                            delarr[dely] := i;
                            break
                        end;
                for i:=delarr[dely]+1 to ListBox1.Items.Count-1 do
                    if ListBox1.Selected[i] then
                        begin
                            nottrue := nottrue + ' and ' + ListBox1.Items[i];
                            dely := dely + 1;
                            delarr[dely] := i;
                        end;
                result := 'not_true(' + Edit1.Text + ',' + nottrue + ')';
                for i := dely downto 1 do
                    begin
                        ListBox1.Items.Delete(delarr[i]);
                        ListBox2.Items.Delete(delarr[i]);
                    end;
                ListBox1.Items.Add(result);
            end
        end;
    end;

    procedure TDefineFactList.BitBtn5Click(Sender: TObject);
```

```
var i :integer;
begin
  ListBox1.Items.Clear;
  ListBox2.Items.Clear;
  for i:=1 to FactNum do
  begin
    ListBox1.Items.Add(FactArr[i]);
    ListBox2.Items.Add(FactName[i]);
  end
end;

procedure TDefineFactList.FormShow(Sender: TObject);
begin
  ListBox1.Items.Clear;
  ListBox2.Items.Clear;
  Edit1.Text := '';
  FactNum := 0;
end;

procedure TDefineFactList.BitBtn6Click(Sender: TObject);
var i,j,s,q : integer;
begin
  RemoveFact.Caption := 'Remove A Fact';
  RemoveFact.Hint := Hint;
  RemoveFact.ListBox1.Items.Clear;
  for i:=0 to ListBox1.Items.Count-1 do
    for j:=1 to FactNum do
      if ListBox1.Items[i] = FactArr[j]
      then RemoveFact.ListBox1.Items.Add(FactArr[j]);
  RemoveFact.ShowModal;
  if RemoveFact.Removed <> '' then
  begin
    for i:=0 to ListBox1.Items.Count-1 do
      if ListBox1.Items[i] = RemoveFact.Removed
      then begin
        for s:=1 to FactNum do
          if ListBox1.Items[i] = FactArr[s]
          then begin
            for q:=s+1 to FactNum do
              FactArr[q-1] := FactArr[q];
              FactNum := FactNum - 1;
            break
          end;
          ListBox1.Items.Delete(i);
          ListBox2.Items.Delete(i);
          break
        end
      end
    end
  end
end;
```

```
end
end;

procedure TDefineFactList.BitBtn3Click(Sender: TObject);
begin
  AddFact.Hint := Hint;
  AddFact.ShowModal;
  if AddFact.Added <> '' then
  begin
    ListBox1.Items.Add(AddFact.Added);
    ListBox2.Items.Add(AddFact.TheName);
    FactNum := FactNum + 1;
    FactArr[FactNum] := AddFact.Added;
    FactName[FactNum] := AddFact.TheName;
  end
end;

end.
{-----}

unit Sss_u10;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TRemoveFact = class(TForm)
    ListBox1: TListBox;
    procedure ListBox1Db1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Removed : string;
  end;

var
  RemoveFact: TRemoveFact;

implementation

{$R *.DFM}
```

```
procedure TRemoveFact.ListBox1DblClick(Sender: TObject);
begin
    Removed := ListBox1.Items[ListBox1.ItemIndex];
    Close;
end;

procedure TRemoveFact.FormShow(Sender: TObject);
begin
    Removed := '';
end;

end.

{-----}

unit Sss_u11;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, ExtCtrls, Buttons, DB, DBTables, Sss_u13, Sss_u6;

type
    TAddFact = class(TForm)
        Bevel1: TBevel;
        Bevel2: TBevel;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        Label4: TLabel;
        Label5: TLabel;
        Edit3: TEdit;
        Edit4: TEdit;
        BitBtn1: TBitBtn;
        Label7: TLabel;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        RadioGroup1: TRadioGroup;
        RadioButton2: TRadioButton;
        RadioButton1: TRadioButton;
        Query1: TQuery;
    end;
```

```
Timer1: TTimer;
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure Edit4Change(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
  Added : string;
  TheName : string;
  ParamList : string;
end;

var
  AddFact: TAddFact;

implementation

{$R *.DFM}

procedure TAddFact.RadioButton2Click(Sender: TObject);
begin
  Edit1.Enabled := False;
  Edit2.Enabled := False;
  Edit3.Enabled := True;
  Edit4.Enabled := True;
  Edit3.SetFocus;
end;

procedure TAddFact.RadioButton1Click(Sender: TObject);
begin
  Edit1.Enabled := True;
  Edit2.Enabled := True;
  Edit3.Enabled := False;
  Edit4.Enabled := False;
  Edit1.SetFocus;
end;

procedure TAddFact.BitBtn3Click(Sender: TObject);
begin
  Added := '';
  Close;
```

```
end;

procedure TAddFact.BitBtn2Click(Sender: TObject);
begin
  if RadioButton1.Checked then
  begin
    Added := 'evaluate(' + Edit1.Text + ',' + Edit2.Text + ')';
    TheName := 'evaluate';
    Close;
  end;
  if RadioButton2.Checked then
  begin
    Added := Edit4.Text + '(' + Edit3.Text + ',' + ParamList + ')';
    TheName := Edit4.Text;
    Close;
  end
end;

procedure TAddFact.BitBtn1Click(Sender: TObject);
var Reference: TControlClass;
    Instance1, Instance2: TControl;
    Text: array[0..255] of Char;
    XPos, YPos, InstNum, i : integer;
    InstArr1 : array [1..15] of TControl;
    InstArr2 : array [1..15] of string;
    okok : boolean;
begin
  Fields.Hint := Hint;
  XPos := 140;
  YPos := 30;
  InstNum := 0;
  Query1.Close;
  Query1.Params[0].AsString := Edit4.Text;
  Query1.Params[1].AsString := Hint;
  Query1.Open;
  if (Query1.EOF)
  then begin
    ErrorBox.Label1.Caption := 'No Such Fact Defined ..';
    ErrorBox.ShowModal;
  end
  else begin
    while not(Query1.EOF) do
    begin
      Reference := TControlClass(TObject(TEdit));
      Instance1 := Reference.Create(Self);
      Instance1.Parent := Fields;
      Instance1.Left := XPos;
```

```

Instance1.Top := YPos;
if Instance1 is TWinControl then
    SendMessage(TWinControl(Instance1).Handle, WM_SETTEXT, 0,
        Longint(StrPCopy(Text, '')));
InstNum := InstNum + 1;
InstArr1[InstNum] := Instance1;
Reference := TControlClass(TObject(TEdit));
Instance2 := Reference.Create(Self);
Instance2.Parent := Fields;
Instance2.Left := XPos - 130;
Instance2.Top := YPos;
Instance2.Enabled := False;
if Instance2 is TWinControl then
    SendMessage(TWinControl(Instance2).Handle, WM_SETTEXT, 0,
        Longint(StrPCopy(Text, Query1.Fields[0].AsString)));
    YPos := YPos + 20;
InstArr2[InstNum] :=Query1.Fields[0].AsString;
Query1.Next;
end;
Fields.Height := YPos+100;
Fields.ShowModal;
okok:=true;
for i:=1 to InstNum do
    if TEdit(InstArr1[InstNum]).Text = '' then okok:=false;
if not(okok)
then begin
    ErrorBox.Label1.Caption := 'Parameter Not Specified';
    ErrorBox.ShowModal;
    ParamList := '';
end
else begin
    ParamList :=InstArr2[1]+' '+TEdit(InstArr1[1]).Text;
    for i:=2 to InstNum do
        ParamList := ParamList + ',' +
            InstArr2[i]+' '+TEdit(InstArr1[i]).Text;
    end
end
end

end;

procedure TAddFact.Edit4Change(Sender: TObject);
begin
    if (Edit4.Text = '') or (Edit4.Text[1] in ['0'..'9'])
    then BitBtn1.Enabled := False
    else BitBtn1.Enabled := True;
end;

```

```
procedure TAddFact.FormShow(Sender: TObject);
begin
  Edit1.Text := '';
  Edit2.Text := '';
  Edit3.Text := '';
  Edit4.Text := '';
  BitBtn1.Enabled := False;
  if RadioButton1.Checked then
  begin
    Edit1.Enabled := True;
    Edit2.Enabled := True;
    Edit3.Enabled := False;
    Edit4.Enabled := False;
  end
  else
  begin
    Edit1.Enabled := False;
    Edit2.Enabled := False;
    Edit3.Enabled := True;
    Edit4.Enabled := True;
  end
end;

procedure TAddFact.Timer1Timer(Sender: TObject);
begin
  if RadioButton1.Checked then
    if (Edit1.Text = '') or (Edit2.Text = '')
    then BitBtn2.Enabled := False
    else BitBtn2.Enabled := True;
  if RadioButton2.Checked then
    if (Edit3.Text = '') or (Edit4.Text = '') or (ParamList = '')
    then BitBtn2.Enabled := False
    else BitBtn2.Enabled := True
end;

end.

{-----}

{ Form Template - Source and Destination Choices Lists }
unit Sss_u12;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Buttons,
  StdCtrls, DBCtrls, DB, DBTables, ExtCtrls;
```

```
type
  TContextAdd = class(TForm)
    OKBtn: TBitBtn;
    DstList: TListBox;
    SrcLabel: TLabel;
    DstLabel: TLabel;
    IncludeBtn: TSpeedButton;
    ExcludeBtn: TSpeedButton;
    Query1: TQuery;
    SrcList: TListBox;
    Timer1: TTimer;
    procedure CancelBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure IncludeBtnClick(Sender: TObject);
    procedure ExcludeBtnClick(Sender: TObject);
    procedure SrcListClick(Sender: TObject);
    procedure DstListClick(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Ok : boolean;
  end;

var
  ContextAdd: TContextAdd;

implementation

{$R *.DFM}

procedure TContextAdd.CancelBtnClick(Sender: TObject);
begin
  Ok := False;
  Close;
end;

procedure TContextAdd.OKBtnClick(Sender: TObject);
begin
  Ok := True;
  Close;
end;

procedure TContextAdd.FormShow(Sender: TObject);
var flag : boolean;
```

```
        i : integer;
begin
    Query1.Close;
    Query1.Params[0].AsString := Hint;
    Query1.Open;
    while not(Query1.EOF) do
    begin
        flag := true;
        for i:=0 to DstList.Items.Count-1 do
            if Query1.Fields[0].AsString = DstList.Items[i] then flag := false;
        if flag then SrcList.Items.Add(Query1.Fields[0].AsString);
        Query1.Next;
        end;
    end;

    procedure TContextAdd.IncludeBtnClick(Sender: TObject);
    var Item : string;
    begin
        Item := SrcList.Items[SrcList.ItemIndex];
        SrcList.Items.Delete(SrcList.ItemIndex);
        DstList.Items.Add(Item);
    end;

    procedure TContextAdd.ExcludeBtnClick(Sender: TObject);
    var Item : string;
    begin
        Item := DstList.Items[DstList.ItemIndex];
        DstList.Items.Delete(DstList.ItemIndex);
        SrcList.Items.Add(Item);
    end;

    procedure TContextAdd.SrcListClick(Sender: TObject);
    begin
        IncludeBtn.Enabled := True;
    end;

    procedure TContextAdd.DstListClick(Sender: TObject);
    begin
        ExcludeBtn.Enabled := True;
    end;

    procedure TContextAdd.Timer1Timer(Sender: TObject);
    begin
        if SrcList.Items.Count = 0 then IncludeBtn.Enabled := False;
        if DstList.Items.Count = 0 then ExcludeBtn.Enabled := False
    end;
end;
```

```
end.

{-----}

unit Sss_u13;

interface

uses

  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons;

type

  TFields = class(TForm)
    BitBtn1: TBitBtn;
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var

  Fields: TFields;

implementation

{$R *.DFM}

procedure TFields.FormShow(Sender: TObject);
begin
  BitBtn1.Left := (Fields.Width DIV 2) - 50;
  BitBtn1.Top := Fields.Height - 70;
end;

end.

{-----}

unit Sss_u14;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;
```

```
type
  TAboutBox = class(TForm)
    Panell1: TPanel;
    OKButton: TBitBtn;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutBox: TAboutBox;

implementation

{$R *.DFM}

end.

{-----}

unit Sss_u15;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, Buttons, Sss_U5, DB, DBTables;

type
  TDefineFact = class(TForm)
    Edit1: TEdit;
    Label2: TLabel;
    ListBox1: TListBox;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Timer1: TTimer;
    Memo1: TMemo;
    Query1: TQuery;
    Table1: TTable;
```

```
Table2: TTable;
procedure Timer1Timer(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
  FactStr : string;
end;

var
  DefineFact: TDefineFact;

implementation

{$R *.DFM}

procedure TDefineFact.Timer1Timer(Sender: TObject);
begin
  if (Edit1.Text='') or (Edit1.Text[1] in ['0'..'9']) or
    (ListBox1.Items.Count=0)
  then BitBtn3.Enabled := False
  else BitBtn3.Enabled := True;
  if(ListBox1.Items.Count=0)
  then BitBtn2.Enabled := False
  else BitBtn2.Enabled := True;
end;

procedure TDefineFact.BitBtn1Click(Sender: TObject);
begin
  DefineVariable.Caption := 'Add A Field';
  DefineVariable.ShowModal;
  if DefineVariable.Variable <> '' then
    ListBox1.Items.Add(DefineVariable.Variable);
end;

procedure TDefineFact.BitBtn2Click(Sender: TObject);
var i : integer;
begin
  for i:=0 to ListBox1.Items.Count-1 do
    if ListBox1.Selected[i] then
      begin
        ListBox1.Items.Delete(i);
```

```
        break;
    end
end;

procedure TDefineFact.BitBtn4Click(Sender: TObject);
begin
    FactStr := '';
    Close;
end;

procedure TDefineFact.FormShow(Sender: TObject);
begin
    FactStr := '';
    Mem0.Lines.Clear;
    ListBox1.Items.Clear;
    Edit1.Text := '';
end;

procedure TDefineFact.BitBtn3Click(Sender: TObject);
var i,Id:integer;
begin
    FactStr := '?- literalize(' + Edit1.Text + '(';
    for i := 0 to ListBox1.Items.Count-1 do
        begin
            if i=ListBox1.Items.Count-1
            then FactStr := FactStr + ListBox1.Items[i] + ')).'
            else FactStr := FactStr + ListBox1.Items[i] + ','
        end;
    Mem0.Lines.Clear;
    Mem0.Lines.Add(FactStr);
    Table1.Append;
    Table1.FieldName('Type').AsString := 'fact';
    Table1.FieldName('String').Assign(Mem0.Lines);
    Table1.FieldName('Name').AsString := Edit1.Text;
    Table1.FieldName('Project').AsString := Hint;
    Table1.Post;
    Query1.Close;
    Query1.Params[0].AsString := Edit1.Text;
    Query1.Params[1].AsString := Hint;
    Query1.Open;
    Id := Query1.Fields[0].AsInteger;
    for i := 0 to ListBox1.Items.Count-1 do
        begin
            Table2.Append;
            Table2.FieldName('ID').AsInteger := Id;
            Table2.FieldName('Field').AsString := ListBox1.Items[i];
            Table2.Post;
        end;
    end;
end;
```

```
    end;
    Close
end;

end.

{-----}

unit Sss_u16;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Sss_U15, Sss_U17, DB, DBTables;

type
    TSelectF = class(TForm)
        Button1: TButton;
        Button2: TButton;
        Mem01: TMem0;
        Table1: TTable;
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        RetRet : string;
    end;

var
    SelectF: TSelectF;

implementation

{$R *.DFM}

procedure TSelectF.Button1Click(Sender: TObject);
begin
    DefineFact.Hint := Hint;
    DefineFact.ShowModal;
    Close;
end;

procedure TSelectF.Button2Click(Sender: TObject);
begin
```

```
MakeFact.Hint := Hint;
MakeFact.ShowModal;
if MakeFact.MakeStr <> '' then
begin
  MakeFact.MakeStr := '?- ' + MakeFact.MakeStr + '.';
  Memol.Lines.Clear;
  Memol.Lines.Add(MakeFact.MakeStr);
  Table1.Append;
  Table1.FieldName('Type').AsString := 'made_fact';
  Table1.FieldName('String').Assign(Memol.Lines);
  Table1.FieldName('Name').AsString := MakeFact.Edit1.Text;
  Table1.FieldName('Project').AsString := Hint;
  Table1.Post;
end;
Close;
end;

end.
{-----}

unit Sss_ul7;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, DB,
  DBTables, Sss_u6, Sss_ul3;

type
  TMakeFact = class(TForm)
    Edit1: TEdit;
    Label2: TLabel;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Edit2: TEdit;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Timer1: TTimer;
    Query1: TQuery;
    Memol: TMemo;
    Table1: TTable;
    procedure BitBtn1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
```

```
private
  { Private declarations }
public
  { Public declarations }
  MakeStr : string;
end;

var
  MakeFact: TMakeFact;
  MakeList : string;

implementation

{$R *.DFM}

procedure TMakeFact.BitBtn1Click(Sender: TObject);
var Reference: TControlClass;
    Instance1, Instance2: TControl;
    Text: array[0..255] of Char;
    XPos, YPos, InstNum, i : integer;
    InstArr1 : array [1..15] of TControl;
    InstArr2 : array [1..15] of string;
    okok : boolean;
begin
  Fields.Hint := Hint;
  XPos := 140;
  YPos := 30;
  InstNum := 0;
  Query1.Close;
  Query1.Params[0].AsString := Edit1.Text;
  Query1.Params[1].AsString := Hint;
  Query1.Open;
  if (Query1.EOF)
  then begin
    ErrorBox.Label1.Caption := 'No Such Fact Defined ..';
    ErrorBox.ShowModal;
  end
  else begin
    while not(Query1.EOF) do
    begin
      Reference := TControlClass(TObject(TEdit));
      Instance1 := Reference.Create(Self);
      Instance1.Parent := Fields;
      Instance1.Left := XPos;
      Instance1.Top := YPos;
      if Instance1 is TWinControl then
        SendMessage(TWinControl(Instance1).Handle, WM_SETTEXT, 0,
```

```

        Longint(StrPCopy(Text, ''));
InstNum := InstNum + 1;
InstArr1[InstNum] := Instance1;
Reference := TControlClass(TObject(TEdit));
Instance2 := Reference.Create(Self);
Instance2.Parent := Fields;
Instance2.Left := XPos - 130;
Instance2.Top := YPos;
Instance2.Enabled := False;
if Instance2 is TWinControl then
    SendMessage(TWinControl(Instance2).Handle, WM_SETTEXT, 0,
        Longint(StrPCopy(Text, Query1.Fields[0].AsString)));
    YPos := YPos + 20;
InstArr2[InstNum] := Query1.Fields[0].AsString;
Query1.Next;
end;
Fields.Height := YPos+100;
Fields.ShowModal;
okok:=true;
for i:=1 to InstNum do
    if TEdit(InstArr1[InstNum]).Text = '' then okok:=false;
if not(okok)
then begin
    ErrorBox.Label1.Caption := 'Parameter Not Specified';
    ErrorBox.ShowModal;
    MakeList := '';
end
else begin
    MakeList :=InstArr2[1]+' '+TEdit(InstArr1[1]).Text;
    for i:=2 to InstNum do
        MakeList := MakeList + ',' +
            InstArr2[i]+' '+TEdit(InstArr1[i]).Text;
    end
end
end;
procedure TMakeFact.Timer1Timer(Sender: TObject);
begin
    if (Edit1.Text = '') or (Edit1.Text[1] in ['0'..'9'])
    then BitBtn1.Enabled := False
    else BitBtn1.Enabled := True;
    if (Edit1.Text = '') or (Edit2.Text = '') or (Edit1.Text[1] in
['0'..'9'])
    or (MakeList = '')
    then BitBtn3.Enabled := False
    else BitBtn3.Enabled := True;
end;

```

```
procedure TMakeFact.BitBtn4Click(Sender: TObject);
begin
  MakeStr := '';
  Close;
end;

procedure TMakeFact.FormShow(Sender: TObject);
begin
  MakeStr := '';
  MakeList := '';
  Mem01.Lines.Clear;
  Edit1.Text := '';
  Edit2.Text := '';
end;

procedure TMakeFact.BitBtn3Click(Sender: TObject);
begin
  MakeStr := 'make(' + Edit1.Text + '(' + Edit2.Text + ','
    + MakeList + ')';
  Close;
end;

end.

{-----}

unit Sss_u18;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Buttons, Sss_u5, DB,
  DBTables, Sss_u25;

type
  TEval = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    BitBtn2: TBitBtn;
    Bevel2: TBevel;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
```

```
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    Label3: TLabel;
    Timer1: TTimer;
    Edit1: TEdit;
    Memo1: TMemo;
    Table1: TTable;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
    Evaluation : string;
end;

var
    Eval: TEval;

implementation

{$R *.DFM}

procedure TEval.BitBtn1Click(Sender: TObject);
begin
    ValueEnt.ShowModal;
    if ValueEnt.Variable <> '' then
        ListBox1.Items.Add(ValueEnt.Variable);
end;

procedure TEval.BitBtn3Click(Sender: TObject);
begin
    ValueEnt.ShowModal;
    if ValueEnt.Variable <> '' then
        ListBox2.Items.Add(ValueEnt.Variable);
end;

procedure TEval.BitBtn2Click(Sender: TObject);
var i:integer;
begin
    for i:=0 to ListBox1.Items.Count-1 do
```

```
    if ListBox1.Selected[i] then
    begin
        ListBox1.Items.Delete(i);
        break;
    end
end;

procedure TEval.BitBtn4Click(Sender: TObject);
var i:integer;
begin
    for i:=0 to ListBox2.Items.Count-1 do
        if ListBox2.Selected[i] then
            begin
                ListBox2.Items.Delete(i);
                break;
            end
        end
    end;
end;

procedure TEval.Timer1Timer(Sender: TObject);
begin
    if (ListBox1.Items.Count = 0) or
        (ListBox2.Items.Count = 0) or
        (Edit1.Text = '') or (Edit1.Text[1] in ['0'..'9'])
    then BitBtn5.Enabled := False
    else BitBtn5.Enabled := True;
    if (ListBox1.Items.Count = 0)
    then BitBtn2.Enabled := False
    else BitBtn2.Enabled := True;
    if (ListBox2.Items.Count = 0)
    then BitBtn4.Enabled := False
    else BitBtn4.Enabled := True;
end;

procedure TEval.BitBtn5Click(Sender: TObject);
var i : integer;
begin
    Evaluation := Edit1.Text + '(';
    for i:=0 to ListBox1.Items.Count-1 do
        if i=ListBox1.Items.Count-1
            then Evaluation := Evaluation + ListBox1.Items[i]
            else Evaluation := Evaluation + ListBox1.Items[i]+',';
    Evaluation := Evaluation + ') :- ';
    for i:=0 to ListBox2.Items.Count-1 do
        if i=ListBox2.Items.Count-1
            then Evaluation := Evaluation + '(' + ListBox2.Items[i] + ') .'
            else Evaluation := Evaluation + '(' + ListBox2.Items[i] + ') ' + ',';
end;
```

```
Memol.Lines.Clear;
Memol.Lines.Add(Evaluation);
Table1.Append;
Table1.FieldName('Type').AsString := 'evaluation';
Table1.FieldName('String').Assign(Memol.Lines);
Table1.FieldName('Name').AsString := Edit1.Text;
Table1.FieldName('Project').AsString := Hint;
Table1.Post;
Close;
end;

procedure TEval.BitBtn6Click(Sender: TObject);
begin
Evaluation := '';
Close;
end;

procedure TEval.FormShow(Sender: TObject);
begin
ListBox1.Items.Clear;
ListBox2.Items.Clear;
Edit1.Text := '';
end;

end.

{-----}

unit Sss_u19;

interface

uses

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs, ExtCtrls, DB, DBTables, StdCtrls, Buttons;

type

TInside = class(TForm)
Listbox1: TListBox;
Listbox2: TListBox;
Listbox3: TListBox;
Listbox4: TListBox;
Listbox5: TListBox;
Listbox6: TListBox;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
```

```
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Table1: TTable;
    Label7: TLabel;
    ListBox7: TListBox;
    BitBtn7: TBitBtn;
    Query1: TQuery;
    Query2: TQuery;
    Query3: TQuery;
    Query4: TQuery;
    Query5: TQuery;
    Query6: TQuery;
    Query7: TQuery;
    Timer1: TTimer;
    ListBox8: TListBox;
    Label8: TLabel;
    Query8: TQuery;
    BitBtn8: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn8Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Inside: TInside;

implementation

{$R *.DFM}
```

```
procedure TInside.BitBtn1Click(Sender: TObject);
var i:integer;
begin
  for i:=0 to ListBox1.Items.Count-1 do
    if ListBox1.Selected[i] then
      begin
        Query1.Close;
        Query1.Params[0].AsString := ListBox1.Items[i];
        Query1.Params[1].AsString := Hint;
        Query1.ExecSQL;
        ListBox1.Items.Delete(i);
        break;
      end;
  end;
end;

procedure TInside.BitBtn2Click(Sender: TObject);
var i:integer;
begin
  for i:=0 to ListBox2.Items.Count-1 do
    if ListBox2.Selected[i] then
      begin
        Query2.Close;
        Query2.Params[0].AsString := ListBox2.Items[i];
        Query2.Params[1].AsString := Hint;
        Query2.ExecSQL;
        ListBox2.Items.Delete(i);
        break;
      end;
  end;
end;

procedure TInside.BitBtn3Click(Sender: TObject);
var i:integer;
begin
  for i:=0 to ListBox3.Items.Count-1 do
    if ListBox3.Selected[i] then
      begin
        Query3.Close;
        Query3.Params[0].AsString := ListBox3.Items[i];
        Query3.Params[1].AsString := Hint;
        Query3.ExecSQL;
        ListBox3.Items.Delete(i);
        break;
      end;
  end;
end;

procedure TInside.BitBtn4Click(Sender: TObject);
var i:integer;
```

```
begin
  for i:=0 to ListBox4.Items.Count-1 do
    if ListBox4.Selected[i] then
      begin
        Query4.Close;
        Query4.Params[0].AsString := ListBox4.Items[i];
        Query4.Params[1].AsString := Hint;
        Query4.ExecSQL;
        ListBox4.Items.Delete(i);
        break;
      end
    end;

  procedure TInside.BitBtn5Click(Sender: TObject);
  var i:integer;
  begin
    for i:=0 to ListBox5.Items.Count-1 do
      if ListBox5.Selected[i] then
        begin
          Query5.Close;
          Query5.Params[0].AsString := ListBox5.Items[i];
          Query5.Params[1].AsString := Hint;
          Query5.ExecSQL;
          ListBox5.Items.Delete(i);
          break;
        end
      end;

  procedure TInside.BitBtn6Click(Sender: TObject);
  var i:integer;
  begin
    for i:=0 to ListBox6.Items.Count-1 do
      if ListBox6.Selected[i] then
        begin
          Query6.Close;
          Query6.Params[0].AsString := ListBox6.Items[i];
          Query6.Params[1].AsString := Hint;
          Query6.ExecSQL;
          ListBox6.Items.Delete(i);
          break;
        end
      end;

  procedure TInside.BitBtn7Click(Sender: TObject);
  var i:integer;
  begin
    for i:=0 to ListBox7.Items.Count-1 do
```

```
    if ListBox7.Selected[i] then
    begin
        Query7.Close;
        Query7.Params[0].AsString := ListBox7.Items[i];
        Query7.Params[1].AsString := Hint;
        Query7.ExecSQL;
        ListBox7.Items.Delete(i);
        break;
    end
end;

procedure TInside.Timer1Timer(Sender: TObject);
begin
    ListBox1.Items.Clear;
    ListBox2.Items.Clear;
    ListBox3.Items.Clear;
    ListBox4.Items.Clear;
    ListBox5.Items.Clear;
    ListBox6.Items.Clear;
    ListBox7.Items.Clear;
    ListBox8.Items.Clear;
    Table1.First;
    while not(Table1.EOF) do
    begin
        if Table1.FieldName('Project').AsString = Hint then
        begin
            if Table1.FieldName('Type').AsString = 'context'
                then ListBox1.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'fail_condition'
                then ListBox2.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'end_goal'
                then ListBox3.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'rule'
                then ListBox4.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'evaluation'
                then ListBox5.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'fact'
                then ListBox6.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'made_fact'
                then ListBox7.Items.Add(Table1.FieldName('Name').AsString);
            if Table1.FieldName('Type').AsString = 'add_context'
                then ListBox8.Items.Add(Table1.FieldName('Name').AsString);
        end;
        Table1.Next;
    end
end;
end;
```

```
procedure TInside.BitBtn8Click(Sender: TObject);
var i:integer;
begin
  for i:=0 to ListBox8.Items.Count-1 do
    if ListBox8.Selected[i] then
      begin
        Query8.Close;
        Query8.Params[0].AsString := ListBox8.Items[i];
        Query8.Params[1].AsString := Hint;
        Query8.ExecSQL;
        ListBox8.Items.Delete(i);
        break;
      end;
  end;
end.

{-----}
unit Sss_u20;

interface

uses

  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons,Sss_ul2,Sss_u9, ExtCtrls,Sss_u22;

type
  TDefineRule = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn5: TBitBtn;
    Timer1: TTimer;
    ListBox1: TListBox;
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
    RuleList : string;
end;

var
    DefineRule: TDefineRule;
    ConArr : Array [1..25] of string;
    ContextNum : integer;
    Lefted,Righted,Conted,Facted : Boolean;

implementation

{$R *.DFM}

procedure TDefineRule.BitBtn3Click(Sender: TObject);
var tempstr : string;
    i : integer;
begin
    tempstr := ConArr[1];
    for i:=2 to ContextNum do
        tempstr := tempstr + ',' + ConArr[i];
    tempstr := 'rule(' + Edit1.Text + ',' + tempstr + ',' +
        DefineFactList.FactList + ' --> '
        + RHS.RhsList + ').';
    RuleList := tempstr;
    Close;
end;

procedure TDefineRule.BitBtn4Click(Sender: TObject);
begin
    RuleList := '';
    Close;
end;

procedure TDefineRule.BitBtn1Click(Sender: TObject);
var i:integer;
begin
    ContextAdd.Hint := Hint;
    ContextAdd.SrcList.Items.Clear;
    ContextAdd.DstList.Items.Clear;
    for i:=1 to ContextNum do
        ContextAdd.DstList.Items.Add(ConArr[i]);
    ContextAdd.ShowModal;
    if ContextAdd.Ok and (ContextAdd.DstList.Items.Count>0) then
    begin
        ContextNum := 0;
        for i:=0 to ContextAdd.DstList.Items.Count-1 do
```

```
begin
    ContextNum := ContextNum + 1;
    ConArr[ContextNum] := ContextAdd.DstList.Items[i]
end;
Conted := True;
end
else begin
    Conted := False;
    ContextAdd.DstList.Items.Clear;
    ContextNum := 0;
end
end;

procedure TDefineRule.BitBtn5Click(Sender: TObject);
var i : integer;
begin
    ListBox1.Items.Clear;
    DefineFactList.Hint := Hint;
    DefineFactList.ShowModal;
    if DefineFactList.FactList <> '' then
        begin
            Facted := True;
            Lefted := True;
            for i:=0 to DefineFactList.ListBox2.Items.Count-1 do
                ListBox1.Items.Add(DefineFactList.ListBox2.Items[i]);
            end
        end
    end;

procedure TDefineRule.BitBtn2Click(Sender: TObject);
var i : integer;
begin
    RHS.Hint := Hint;
    RHS.ListBox2.Items.Clear;
    for i:=0 to ListBox1.Items.Count-1 do
        RHS.ListBox2.Items.Add(ListBox1.Items[i]);
    end
    RHS.ShowModal;
    if RHS.RhsList <> '' then
        begin
            Righted := True;
        end
    end;

procedure TDefineRule.FormShow(Sender: TObject);
begin
    ListBox1.Items.Clear;
    Conted := False;
```

```
Lefted := False;
Righted := False;
Edit1.Text := '';
end;

procedure TDefineRule.Timer1Timer(Sender: TObject);
begin
  if Lefted then BitBtn2.Enabled := True
    else BitBtn2.Enabled := False;
  if Conted and Lefted and Righted and (Edit1.Text <> '') and
    not(Edit1.Text[1] in ['0'..'9'])
    then BitBtn3.Enabled := True
    else BitBtn3.Enabled := False;
end;

end.

{-----}
unit Sss_u21;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, DB, DBTables, Sss_u5, Sss_u10;

type
  TSelectC = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Memo1: TMemo;
    Table1: TTable;
    Query1: TQuery;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  SelectC: TSelectC;

implementation

{$R *.DFM}
```

```
procedure TSelectC.Button1Click(Sender: TObject);
var temp : string;
begin
  DefineVariable.Caption := 'Define A Context';
  DefineVariable.ShowModal;
  if DefineVariable.Variable<>' ' then
  begin
    Memo1.Lines.Clear;
    temp := 'context(' + DefineVariable.Variable + ').';
    Memo1.Lines.Add(temp);
    Table1.Append;
    Table1.FieldName('Type').AsString := 'context';
    Table1.FieldName('String').Assign(Memo1.Lines);
    Table1.FieldName('Name').AsString := DefineVariable.Variable;
    Table1.FieldName('Project').AsString := Hint;
    Table1.Post;
  end;
  Close;
end;

procedure TSelectC.Button2Click(Sender: TObject);
var temp : string;
begin
  RemoveFact.ListBox1.Items.Clear;
  Query1.Close;
  Query1.Params[0].AsString := Hint;
  Query1.Open;
  while not(Query1.EOF) do
  begin
    RemoveFact.ListBox1.Items.Add(Query1.Fields[0].AsString);
    Query1.Next;
  end;
  RemoveFact.Caption := 'Add A Context';
  RemoveFact.ShowModal;
  if RemoveFact.Removed <> ' ' then
  begin
    Memo1.Lines.Clear;
    Temp := '?- add_context(' + RemoveFact.Removed + ').';
    Memo1.Lines.Add(Temp);
    Table1.Append;
    Table1.FieldName('Type').AsString := 'add_context';
    Table1.FieldName('String').Assign(Memo1.Lines);
    Table1.FieldName('Name').AsString := RemoveFact.Removed;
    Table1.FieldName('Project').AsString := Hint;
    Table1.Post;
  end;
end;
```

```
    Close;
end;

end.

{-----}

unit Sss_u22;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons,
    ExtCtrls, Sss_u5, Sss_u23, Sss_u24, Sss_ul7,
    DB, DBTables, Sss_ul0;

type
    TRHS = class(TForm)
        ListBox1: TListBox;
        BitBtn1: TBitBtn;
        BitBtn3: TBitBtn;
        BitBtn6: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn4: TBitBtn;
        BitBtn5: TBitBtn;
        BitBtn7: TBitBtn;
        BitBtn8: TBitBtn;
        Timer1: TTimer;
        BitBtn9: TBitBtn;
        Edit1: TEdit;
        ListBox2: TListBox;
        Query1: TQuery;
        procedure BitBtn1Click(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure BitBtn9Click(Sender: TObject);
        procedure BitBtn8Click(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure BitBtn6Click(Sender: TObject);
        procedure BitBtn4Click(Sender: TObject);
        procedure BitBtn7Click(Sender: TObject);
        procedure FormShow(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure BitBtn5Click(Sender: TObject);
    private
        { Private declarations }
    public
```

```
        { Public declarations }
        RhsList : string;
    end;

var
    RHS: TRHS;

implementation

{$R *.DFM}

procedure TRHS.BitBtn1Click(Sender: TObject);
var temp:string;
    i : integer;
begin
    temp := ListBox1.Items[0];
    for i:=1 to ListBox1.Items.Count-1 do
        temp := temp + ' and ' + ListBox1.Items[i];
    RhsList := temp;
    Close;
end;

procedure TRHS.Timer1Timer(Sender: TObject);
var ccc,eee : integer;
begin
    if ListBox1.Items.Count = 0
        then begin
            BitBtn1.Enabled := False;
            BitBtn9.Enabled := False;
        end
        else begin
            BitBtn1.Enabled := True;
            BitBtn9.Enabled := True;
        end;
    Val(Edit1.Text,ccc,eee);
    if (Edit1.Text <> '') and (ccc <= ListBox2.Items.Count) and
        (eee = 0) and (ccc > 0)
        then BitBtn4.Enabled := True
        else BitBtn4.Enabled := False;
end;

procedure TRHS.BitBtn9Click(Sender: TObject);
var i:integer;
begin
    for i:=0 to ListBox1.Items.Count-1 do
        if ListBox1.Selected[i] then
            begin
```

```
ListBox1.Items.Delete(i);
break;
end;
end;

procedure TRHS.BitBtn8Click(Sender: TObject);
begin
  RhsList := '';
  Close;
end;

procedure TRHS.BitBtn3Click(Sender: TObject);
var Temp : string;
begin
  RemoveFact.ListBox1.Items.Clear;
  Query1.Close;
  Query1.Params[0].AsString := Hint;
  Query1.Open;
  while not(Query1.EOF) do
  begin
    RemoveFact.ListBox1.Items.Add(Query1.Fields[0].AsString);
    Query1.Next
  end;
  RemoveFact.Caption := 'Add A Context';
  RemoveFact.ShowModal;
  if RemoveFact.Removed <> '' then
  begin
    Temp := 'add_context(' + RemoveFact.Removed + ')';
    ListBox1.Items.Add(Temp)
  end
end;

procedure TRHS.BitBtn6Click(Sender: TObject);
var Temp : string;
begin
  RemoveFact.ListBox1.Items.Clear;
  Query1.Close;
  Query1.Params[0].AsString := Hint;
  Query1.Open;
  while not(Query1.EOF) do
  begin
    RemoveFact.ListBox1.Items.Add(Query1.Fields[0].AsString);
    Query1.Next
  end;
  RemoveFact.Caption := 'Add A Context';
  RemoveFact.ShowModal;
  if RemoveFact.Removed <> '' then
```

```
begin
  Temp := 'remove_context(' + RemoveFact.Removed + ')';
  ListBox1.Items.Add(Temp)
end
end;

procedure TRHS.BitBtn4Click(Sender: TObject);
var temp : string;
begin
  temp := 'remove(' + Edit1.Text + ')';
  ListBox1.Items.Add(temp);
end;

procedure TRHS.BitBtn7Click(Sender: TObject);
var temp : string;
begin
  WriteLine.Hint := Hint;
  WriteLine.ShowModal;
  if WriteLine.WL <> '' then
  begin
    temp := 'writeline(';
    temp := temp + WriteLine.WL + ')';
    ListBox1.Items.Add(temp)
  end
end;

procedure TRHS.FormShow(Sender: TObject);
begin
  ListBox1.Items.Clear;
  Edit1.Text := '';
end;

procedure TRHS.BitBtn2Click(Sender: TObject);
var temp : string;
    i : integer;
begin
  Modify.Hint := Hint;
  Modify.ListBox1.Items.Clear;
  for i:=0 to ListBox2.Items.Count-1 do
    Modify.ListBox1.Items.Add(ListBox2.Items[i]);
  Modify.ShowModal;
  if Modify.ModifyList <> '' then
  begin
    temp := 'modify(' + Modify.ModifyList + ')';
    ListBox1.Items.Add(temp);
  end
end;
end;
```

```
procedure TRHS.BitBtn5Click(Sender: TObject);
begin
    MakeFact.Hint := Hint;
    MakeFact.ShowModal;
    if MakeFact.MakeStr <> ''
        then ListBox1.Items.Add(MakeFact.MakeStr)
    end;
end;

end.

{-----}

unit Sss_u23;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons, Sss_u5, ExtCtrls;

type
    TWriteLine = class(TForm)
        ListBox1: TListBox;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        BitBtn4: TBitBtn;
        Timer1: TTimer;
        procedure BitBtn4Click(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure FormShow(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        WL : string;
    end;

var
    WriteLine: TWriteLine;

implementation
```

```
{ $R *.DFM }

procedure TWriteLine.BitBtn4Click(Sender: TObject);
begin
  WL := '';
  Close;
end;

procedure TWriteLine.BitBtn3Click(Sender: TObject);
var i : integer;
begin
  WL := '[';
  for i := 0 to ListBox1.Items.Count-1 do
  begin
    if i=ListBox1.Items.Count-1
    then WL := WL + ListBox1.Items[i] + ']'
    else WL := WL + ListBox1.Items[i] + ','
    end;
  Close;
end;

procedure TWriteLine.BitBtn1Click(Sender: TObject);
begin
  DefineVariable.Caption := 'Add A Field';
  DefineVariable.ShowModal;
  if DefineVariable.Variable <> '' then
  ListBox1.Items.Add(DefineVariable.Variable);
end;

procedure TWriteLine.BitBtn2Click(Sender: TObject);
var i : integer;
begin
  for i:=0 to ListBox1.Items.Count-1 do
  if ListBox1.Selected[i] then
  begin
    ListBox1.Items.Delete(i);
    break;
  end
end;
end;

procedure TWriteLine.Timer1Timer(Sender: TObject);
begin
  if ListBox1.Items.Count = 0
  then begin
    BitBtn2.Enabled := False;
    BitBtn3.Enabled := False
  end
end;
```

```
        else begin
            BitBtn2.Enabled := True;
            BitBtn3.Enabled := True;
        end
    end;

end;

procedure TWriteLine.FormShow(Sender: TObject);
begin
    ListBox1.Items.Clear;
end;

end.

{-----}

unit Sss_u24;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, ExtCtrls, StdCtrls, Buttons, DB,
    DBTables, Sss_u13, Sss_u6;

type
    TModify = class(TForm)
        Label1: TLabel;
        Label2: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        Timer1: TTimer;
        BitBtn3: TBitBtn;
        Query1: TQuery;
        ListBox1: TListBox;
        procedure BitBtn2Click(Sender: TObject);
        procedure BitBtn1Click(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure FormShow(Sender: TObject);
    private
        { Private declarations }
    public
```

```
        { Public declarations }
        ModifyList : string;
    end;

var
    Modify: TModify;
    ParamList : string;

implementation

{$R *.DFM}

procedure TModify.BitBtn2Click(Sender: TObject);
begin
    ModifyList := '';
    Close;
end;

procedure TModify.BitBtn1Click(Sender: TObject);
begin
    ModifyList := Edit1.Text + ',' + Edit2.Text + ',' + ParamList;
    Close;
end;

procedure TModify.Timer1Timer(Sender: TObject);
var ccl,eel : integer;
begin
    Val(Edit2.Text,ccl,eel);
    if (Edit1.Text <> '') and
        (Edit2.Text <> '') and
        (ParamList <> '') and (eel = 0)
    then BitBtn1.Enabled := True
    else BitBtn1.Enabled := False;
    if (Edit2.Text <> '') and (eel = 0)
    then BitBtn3.Enabled := True
    else BitBtn3.Enabled := False;
end;

procedure TModify.BitBtn3Click(Sender: TObject);
var Reference: TControlClass;
    Instance1,Instance2: TControl;
    Text: array[0..255] of Char;
    XPos,YPos,InstNum,i : integer;
    InstArr1 : array [1..15] of TControl;
    InstArr2 : array [1..15] of string;
    okok : boolean;
begin
```

```
Fields.Hint := Hint;
XPos := 140;
YPos := 30;
InstNum := 0;
if StrToInt(Edit2.Text)-1 > ListBox1.Items.Count - 1
then begin
    ErrorBox.Labell.Caption := 'No Such Numbered Fact ..';
    ErrorBox.ShowModal;
end
else begin
    Query1.Close;
    Query1.Params[0].AsString := ListBox1.Items[StrToInt(Edit2.Text)-1];
    Query1.Params[1].AsString := Hint;
    Query1.Open;
    if (Query1.EOF)
    then begin
        ErrorBox.Labell.Caption := 'No Such Fact Defined ..';
        ErrorBox.ShowModal;
    end
else begin
    while not(Query1.EOF) do
    begin
        Reference := TControlClass(TObject(TEdit));
        Instance1 := Reference.Create(Self);
        Instance1.Parent := Fields;
        Instance1.Left := XPos;
        Instance1.Top := YPos;
        if Instance1 is TWinControl then
            SendMessage(TWinControl(Instance1).Handle, WM_SETTEXT, 0,
                Longint(StrPCopy(Text, '')));
        InstNum := InstNum + 1;
        InstArr1[InstNum] := Instance1;
        Reference := TControlClass(TObject(TEdit));
        Instance2 := Reference.Create(Self);
        Instance2.Parent := Fields;
        Instance2.Left := XPos - 130;
        Instance2.Top := YPos;
        Instance2.Enabled := False;
        if Instance2 is TWinControl then
            SendMessage(TWinControl(Instance2).Handle, WM_SETTEXT, 0,
                Longint(StrPCopy(Text, Query1.Fields[0].AsString)));
        YPos := YPos + 20;
        InstArr2[InstNum] := Query1.Fields[0].AsString;
        Query1.Next;
    end;
    Fields.Height := YPos+100;
    Fields.ShowModal;
```

```
        okok:=true;
        for i:=1 to InstNum do
            if TEdit(InstArr1[InstNum]).Text = '' then okok:=false;
            if not(okok)
            then begin
                ErrorBox.Label1.Caption := 'Parameter Not Specified';
                ErrorBox.ShowModal;
                ParamList := '';
            end
            else begin
                ParamList :=InstArr2[1]+' '+TEdit(InstArr1[1]).Text;
                for i:=2 to InstNum do
                    ParamList := ParamList + ',' +
                        InstArr2[i]+' '+TEdit(InstArr1[i]).Text;
                end
            end
        end
    end
end;

procedure TModify.FormShow(Sender: TObject);
begin
    Edit1.Text := '';
    Edit2.Text := '';
end;

end.

{-----}

unit Sss_u25;

interface

uses

    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons;

type

    TValueEnt = class(TForm)
        Label1: TLabel;
        Edit1: TEdit;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure Edit1Change(Sender: TObject);
        procedure FormShow(Sender: TObject);
```

```
private
  { Private declarations }
public
  { Public declarations }
  Variable : string;
end;

var
  ValueEnt: TValueEnt;

implementation

{$R *.DFM}

procedure TValueEnt.BitBtn1Click(Sender: TObject);
begin
  Variable := Edit1.Text;
  Close;
end;

procedure TValueEnt.BitBtn2Click(Sender: TObject);
begin
  Variable := '';
  Close;
end;

procedure TValueEnt.Edit1Change(Sender: TObject);
begin
  if Edit1.Text = '' then BitBtn1.Enabled := False
    else BitBtn1.Enabled := True;
end;

procedure TValueEnt.FormShow(Sender: TObject);
begin
  Edit1.Text := '';
  BitBtn1.Enabled := False;
  Variable := '';
end;

end.

{-----}

unit Sss_u26;

interface
```

```
uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, ExtCtrls;

type
    TCopyright = class(TForm)
        Image1: TImage;
        Bevel1: TBevel;
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Copyright: TCopyright;

implementation

{$R *.DFM}

end.

{-----}

unit Sss_u27;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, ExtCtrls, StdCtrls, Sss_u2;

type
    TMain = class(TForm)
        Image1: TImage;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Bevel1: TBevel;
        Timer1: TTimer;
        Bevel2: TBevel;
        Bevel3: TBevel;
        procedure Timer1Timer(Sender: TObject);
    private
        { Private declarations }
    public
```

```
        { Public declarations }
    end;

var
    Main: TMain;

implementation

{$R *.DFM}

procedure TMain.Timer1Timer(Sender: TObject);
begin
    MainMenu.Show;
    Hide;
    Timer1.Enabled := False;
end;
end.
```

APPENDIX C

INSTALLATION GUIDE FOR SSS IDE SYSTEM

Our software product, runs basically on a PC and this PC has to satisfy the requirements which may be specified as follows :

<u>CPU</u>	:	has to be at least Intel 80286.
<u>Memory</u>	:	has to be 2 MB minimum.
<u>Windows Version</u>	:	has to be 3.1 up to 4.0 (i.e. Windows 95)
<u>Disk Space</u>	:	nearly 20 MB of hard disk space is needed.

You can install SSS IDE by its installation program. This product and its installation program are both Windows applications, so you must already have Windows running to install SSS. The installation program creates directories as needed and copies files from the distribution disk to your hard drive. Mainly, drive C is used.

The installation program is largely self-explanatory. The following steps tell you all you need to know to install the product.

To install,

- Start Windows if it is not already running on your computer.
- Insert the installation diskette to your floppy diskette drive.
- Use Program Manager's File / Run (Start / Run at Windows 95) menu command or File Manager (Windows Explorer at Windows 95) to run 'INSTALL.EXE' from the installation diskette.
- Follow the instructions presented by the installation program.

BIBLIOGRAPHY

Aho, A., J Hopcroft, and J. Ullman. 1983. Data Structures and Algorithms. Reading, MA: Addison - Wesley.

Aho, A. V. and S. C. Johnson. 1984. LR Parsing. Computing Surveys 6:2, 99 - 124.

Aho, A. V. and J. D. Ullman. 1973. A Technique For Speeding Up LR(k) Parsers. SIAM J. Computing 2:2, 106-127.

Aho, A. V., J. D. Ullman and R. Sethi. 1986. Compilers Principles, Techniques and Tools. Addison - Wesley.

Barr, A., E. A. Feigenbaum, and P. R. Cohen. 1981. The Handbook of Artificial Intelligence. Los Altos, CA: Morgan Kaufman.

C++ Class Libraries Guide. 1994. Borland International, Inc.

DeRemer, F. 1971. Simple LR(k) grammars. Comm. ACM 14:7, 453-460.

Delphi On-Line Manuals. 1995. Borland International, Inc.

Nilsson, N. J. 1986. Probabilistic logic. Artificial Intelligence 28(1): 71-87.

ObjectWindows Programming Guide. 1992. Borland International, Inc.

Turbo Pascal Programmer's Reference. 1992. Borland International, Inc.

Weiss, S. M. and C. A. Kulikowski. 1984. A Practical Guide to Designing Expert Systems. Towata, NJ: Rowman & Allanheld.

Waterman, D. A. 1986. A Guide to Expert Systems. Reading, MA: Addison - Wesley.

REFERENCES NOT CITED

Bison Parser Generator Reference Guide. 1994. DJ Delorie.

Brownston, L., R. Farrell, E. Kant, and N. Martin. 1985. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Reading, MA: Addison - Wesley.

C++ Class Libraries Guide. 1994. Borland International, Inc.

Chomsky, N. 1957. Syntactic Structures. The Hague: Motion.

ObjectWindows Programming Guide. 1992. Borland International, Inc.

Quine, W. V. 1961. From a Logical Point of View, 2nd ed. New York: Harper.

Shafer, G. and J. Pearl, eds 1990. Readings in Uncertain Reasoning. Los Altos, CA: Morgan Kaufman.