

BUILDING AN EXPERT SYSTEM  
BY INTEGRATING CLIPS WITH VISUAL BASIC

By

YUN HUI LU

Bachelor of Science

Shanghai Post and Telecommunications College

Shanghai, China

1983

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
July, 2000

Oklahoma State University Library

BUILDING AN EXPERT SYSTEM  
BY INTEGRATING CLIPS WITH VISUAL BASIC

Thesis Approved:

*Jacques E. LaFrance*  
Thesis Adviser

*Jane Terry Butler*

*J. Chandler*

*Alfred Sarlozzi*  
Dean of the Graduate College

## ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor Dr. Jacques LaFrance for his counsel, encouragement, trust and friendship throughout my graduate program. Also, my sincere appreciation is extended to the other committee members Dr. John P. Chandler and Dr. Jane Terry Nutter, whose constructive guidance, instruction, and friendship are also invaluable. Completion of this thesis would not have been possible without their diligent supervision.

I would also like to give my special appreciation to my parents, sister and brother-in-law, for their loving encouragement, support and understanding while working on my master degree. This endeavor would not have been possible without their love and belief in my abilities.

Moreover, I wish to express my sincere gratitude to those friends who provided assistance and encouragement for this study, especially to my classmate Xiao Zhen Wang for his precious suggestions, Their friendship made the ending of this study go so smoothly.

Finally, I would like to thank the Department of Computer Science for providing me the opportunity and knowledge during these three years of graduate study.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
Background .....	1
Intent of Study .....	2
Project Phases .....	4
II. LITERATURE REVIEW .....	6
Development of Expert Systems.....	6
CLIPS.....	8
Features of VB Interface .....	10
Dynamic Link Library .....	12
III. DESIGN AND IMPLEMENTATION .....	13
Program Design .....	13
Implementation of the GUI .....	14
Embedded CLIPS within VB5 by DLL .....	17
Facilities of VB GUI .....	20
Implementation of the Rule-based Expert System .....	23
Implementation of the Inference Engine .....	25
Design of Facts and Templates .....	27
Formulation of Rules .....	28
Database Supplement .....	33
IV. SUMMARY AND CONCLUSIONS .....	35
BIBLIOGRAPHY .....	37
APPENDIX .....	39

## TABLE OF FIGURES

Figure	Page
1. Overview of Core Components of the Expert System .....	3
2. Basic Concept of An Expert System Function .....	7
3. Food Guide Pyramid .....	8
4. The Interface of the Expert System in CLIPS Shell .....	10
5. The Main Application Window .....	14
6. A List of Attributes and Methods with VB5 .....	15
7. Attributes and Methods of Personalinfo Object .....	15
8. The Interface for the User to Input Actual Daily Intake .....	19
9. The Interface of the CLIPS Expert System Embedded into VB5 .....	19
10. Structure of a Rule-based Expert System .....	25
11. Nutrition Decision Tree with Multiple Branches .....	26
12. Rules in Agenda with CLIPS .....	29
13. Watch Rules and Facts with CLIPS .....	32
14. Comparison of the Result of Food Servings that the User Provided .....	34

---

## **CHAPTER I**

### **Introduction**

#### **Background**

As a branch of Artificial Intelligence, expert systems are computer systems that emulate the decision-making ability of human experts. They have been used as “expert consultants” to solve real world problems which normally would require a specialized human expert but stem from the shortage of human experts. In addition, expert systems have played an important role in distance education, and to provide expert knowledge readily when and where users need it. Hence, experts will no longer just provide their services in a face-to-face way, but distribute their expert knowledge through the application of expert systems.

Expert systems are developed to improve users’ ease in accessing the expert information without experts present. They imitate the interaction between experts and users so as to identify the requirement of users and the solution of the problems. Originally, users could only interact with the program by using on-line commands to communicate with the knowledge base in a standard expert system environment. These expert system programs could often be well accessed by human experts, but it may be not

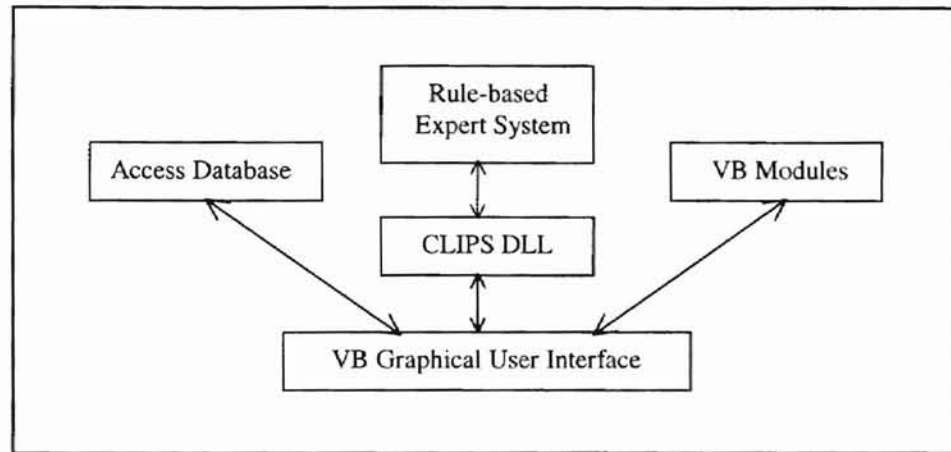
easy for the less-trained users to access or use these programs. In today's environment, with a great variety of computer systems and different operating systems in use, data sharing between computer systems becomes more difficult and costly. How can expert systems help to distribute expertise information to various novice users, to allow them to interact with the knowledge-based systems, and to access the required information in an easy way? In view of this, it is important to establish a link between some efficient graphical user interfaces (GUI) and expert systems in order to support the novice users' access to all expertise information without remembering the unfamiliar textual commands.

### **Intent of study**

The primary objective of this study is to develop an integrated application using an embedded CLIPS rule-based expert system within a Visual Basic application using a customized user interface application. This rule-based expert system has been developed in a Windows environment by using CLIPS (C Language Integrated Production System). It is developed with an example expert system of nutrition assessment for this thesis. The expert system is executed by the inference engine that is built in the CLIPS system as a statement interpreter. So normally, the CLIPS program should be run under the CLIPS shell interface.

The VB application acts like a Windows environment application to enhance the nutrition problem-solving and service-support functions of the example expert system. The VB application is an even-driven program. It is created using VB GUI technology to

achieve the interaction between the expert system and users. Recently, Visual Basic 5.0 has become a strong tool to develop a GUI for different Windows applications. It makes the design of the GUI easy and efficient.



**Figure 1.** Overview of Core Components of the Expert System

A block diagram (Fig. 1) illustrates the configuration of this application.

- The VB GUI is the users' view of the system. It allows the user to supply answers for the expert system. This data is passed to the CLIPS program through the DLL message routing mechanism. The GUI also shows the result of the reasoning session to the user.
- The CLIPS DLL allows information to be exchanged between the VB GUI and the rule-based expert system.
- The CLIPS rule-based expert system contains a set of rules that implement the "match" reasoning strategy to achieve the nutrition assessment.
- The Access database allows the user to access more nutrition information. Therefore, it enhances the knowledge base and facilities of the expert system.
- The VB modules are used to call the CLIPS DLL functions and to connect with the Access database.



Finally, all components are incorporated into a VB interface. Then users are able to interact with the CLIPS inference engine in a Windows-based environment to obtain the desired answers and conclusions. The major accomplishment of this study has been to develop a visual user interface with the CLIPS rule-based expert system so as to make the CLIPS program more efficient and user-friendly.

### **Project Phases**

As mentioned before, the expert system in this study is comprised of three main components, a rule-based expert system, an expert knowledge base and an efficient graphical user interface. The process of developing this expert system is provided as follows:

1. The development of the rule-based expert system employed an expert system development shell CLIPS 6.10. The CLIPS 6.10 compiler with an editor is 595KB in size. Hence, the use of the CLIPS Shell to write expert systems would generally reduce the cost and time of developing software for multiple-platform use. It provides a complete environment consisting of an inference engine, a user interface, an explanation system, and a knowledge base editor for the construction of the rule-based system.
2. The design of an expert system graphical user interface for nutrition assessment has been done by the Visual Basic development tool, VB5.0.
3. The design and set up of a nutrition information database is completed by using a relational database tool Microsoft Access 7. It contains six entities namely grain,

---

vegetable, fruit, meat, fats and oil and milk. The entities possess certain attributes.

These attributes correspond to the fields in the table of the database.

4. The provision of four facilities of this expert system to users are: the interaction between the expert system and users, the evaluation of personal daily intake, the analysis of calories needed for individual, and the acquisition of nutrition knowledge.
5. The testing of the usability and consistency of the GUI has been done accordingly.
6. The integration and the testing of the sample expert system have been done.

## **CHAPTER II**

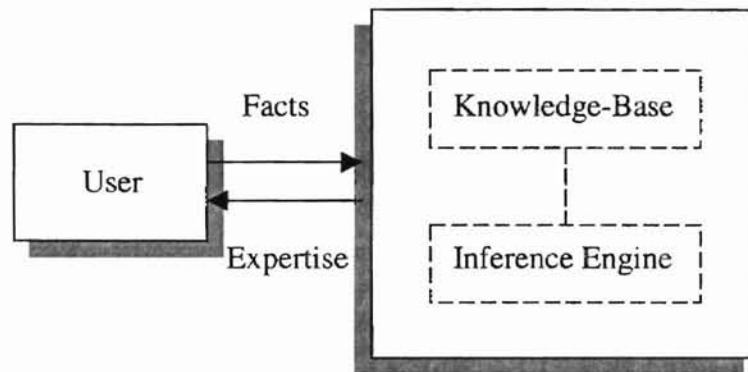
### **Literature Review**

#### **Development of Expert Systems**

An expert system is a computer program. It is used as an Artificial Intelligence (AI) tool to model and aid human decision making in a highly specialized problem domain.

The concept of an expert system was spawned in the early seventies [Waterman, 1986]. In the sixties, AI scientists tried to simulate the complicated process of thinking by finding general methods for solving broad classes of problems. Despite some interesting attempts, this strategy produced no breakthroughs in general-purpose programs. The more classes of problems a single program could handle, the worse it seemed to do on any individual problem. AI scientists decided that there must be another way to make a computer program more efficient in AI fields. The conceptual breakthrough in achieving that was made in the late seventies. The point was that AI programs should focus on some special problem areas that usually require human expertise's knowledge. They should also have lots of high-quality, specific knowledge about the special problem areas. This evolution made the expert systems to a higher platform.

In a normal expert system environment, there are three basic elements, namely the user interface, the inference engine, and the expert knowledge base. The user interface may use menus, natural language or any other style of interactions. The inference engine is used to draw conclusions with both the expert knowledge and data specific to the particular problem being solved (See Fig. 2).

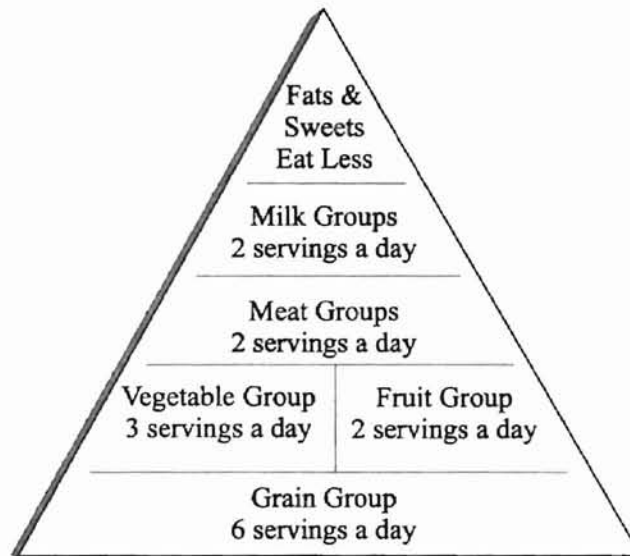


**Figure 2.** Basic Concept of An Expert System Function

Expert systems have been used to solve a wide range of problems in domains such as medicine, mathematics, engineering, geology, computer science, business, law, defense and education. Within each domain, they have been used to solve problems of different types, such as disease diagnosis, chemical analysis, computer system design, geological data interpretation, wine selection, and armored vehicle identification.

In recent years, many expert system applications have been developed in nutrition assessment, consultation and education. According to the references for this study, most of the expert system applications in the nutrition domain need a special knowledge base for emulating expert functions. For the nutrition assessment, the basic dietary standard such as Recommended Dietary allowances (RDA) is issued by the U.S. Federal Government. Another standard is Dietary Recommendations, which is quite different

from the RDA. While RDA deals with specific nutrients, Dietary Recommendations discuss specific foods and food groups that will help individuals meet the RDA. The most recent set of U.S Dietary recommendations is referred to as the Food Guide Pyramid. (See Fig. 3)



**Figure 3.** Food Guide Pyramid

### CLIPS

CLIPS is a productive development and delivery expert system tool introduced by NASA in 1985. CLIPS is written in C for speed and portability, and it uses a powerful pattern matching process called the Rete Algorithm and a forward chaining rule-based language. The algorithm does not iterate over the set of rules, it just processes the changes and updates the elements associated with the patterns. Any compiler that supports the standard Kernighan and Ritchie C language can be used to install CLIPS [Giarratano and Riley, 1989]. The basic elements of a CLIPS program are:

- Fact-list: working memory for data
- Knowledge-base: contains all the rules
- Inference engine: controls overall execution

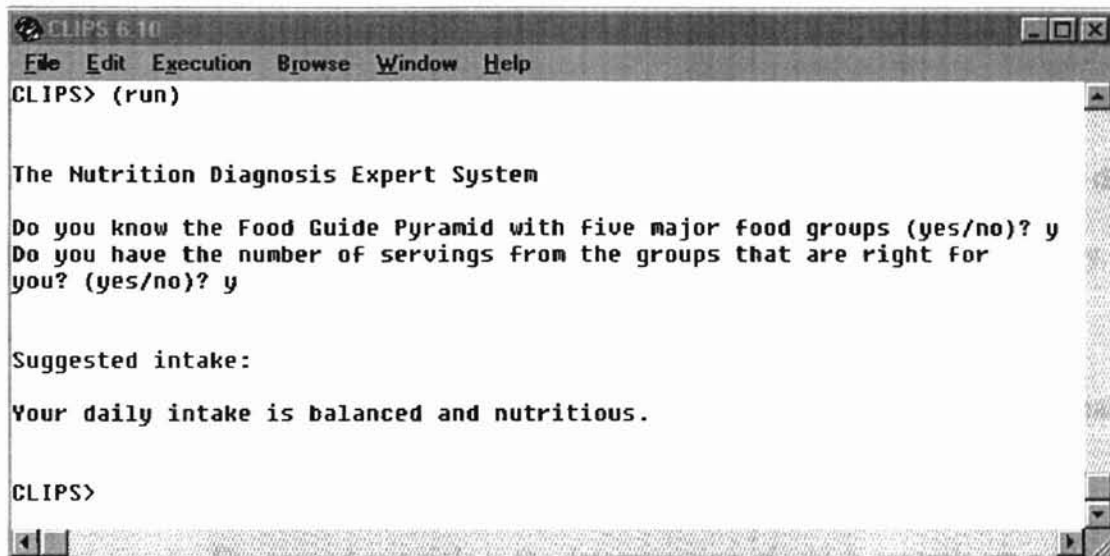
Facts are the first component of a CLIPS program. Facts are the information needed for the decision making process. They are made up of fields that are either a word, string, or number. The first field of a fact is normally used to indicate the type of information stored in the fact, and is called a relation. If some facts are asserted in the fact-list, the inference engine will search the fact-list to match the rules. Then the actions of the rule will be executed and new facts will be provided for next process.

Rules are the second component of a CLIPS system. A rule is divided into the left hand side (LHS) and the right hand side (RHS). The LHS in a rule can be thought of as the IF portion, it contains some matching patterns. The RHS can be thought of as the THEN portion. It contains some actions. A rule can have multiple patterns and actions.

The third component of CLIPS is the inference engine that applies the rules. In an inference structure, rules will be placed into the program's agenda after they have been activated. The activation of the rules is based upon the satisfaction of the pattern matching by various facts. The activated rule will be fired if its patterns are matched by the facts, and the actions of that rule will be executed, therefore new facts are asserted. The inference process will continue until the last rule is reached.

The generic interface of a CLIPS program is a simple, interactive, text-oriented, and command prompt interface. Similar to a DOS or UNIX environment, CLIPS has its text-oriented property, while the text may be rapid, it would appear boring to the users. Also, the representation of numbers on the screen would not be easily interpreted by the users if

not accompanied with a visual representation of the output. In addition, users would need to have some minimal knowledge of using CLIPS before they could meaningfully utilize the software, especially when CLIPS requires specific commands to run. This knowledge would not be known by the general public other than the experts (See Fig. 4).



**Figure 4.** The Interface of the Expert System in CLIPS Shell

Therefore, subsequent enhancements to CLIPS are extended to incorporate CLIPS with a widely-adopted graphic representation program such as Windows based Visual Basic 5.0. This integration will provide for a more convenient tool for the general public using the CLIPS expert system application.

### Features of VB Interface

Visual basic is designed to make user-friendly programs easier to develop. The sequence of procedures executed in the VB program is controlled by the "event" that the

user initiates [Schneider 1998]. Objects in Visual Basic recognize events like mouse movements, and respond to these events depending on the instructions to be written.

VB provides two features different from other programming tools. Instead of writing many sequences of function calls, the VB interface can be designed by dragging controls onto a form. The program displays the Windows style interface with command buttons, text boxes and other objects that will initiate actions when the user moves the mouse, clicks, and strikes keys. These objects provide a visual guide to what the program can do. The well-designed VB user interface insulates the user from the underlying technology, making it easier to perform the intended task.

To design a Visual Basic interface program, there are several important conventional principles to ensure a good user interface. First, an important principle is to keep an interface simple. The second principle is the determination of which controls are needed. It can be achieved by determining how the input will be obtained and how the output will be displayed, then creating objects to receive the input and display the output. Also some appropriate command buttons need to be created to allow the user to control the program. Meanwhile, it is necessary to ensure the more important elements are readily apparent to the user. Third, the consistencies between these controls are very important to usability. Fourth, the interface should prompt the user for missing information before continuing the input session. Lastly, it should have a standard Windows look and feel [Wolke 1997].



## Dynamic Link Library

The Dynamic Link Library (DLL) contains executable code and data bound to a program at load time or run time. The code and data in a DLL can be shared by several applications simultaneously. It has the following advantages:

1. Applications would link to this code library, thus saving greatly on duplication of effort and storage space.
2. Applications that used the same DLL system would behave identically.
3. If a problem arose, or a new feature was desired, the DLL would only need to be edited/written once to benefit all related software. In this sense, the DLL system is a weak version of the object-programming paradigm.

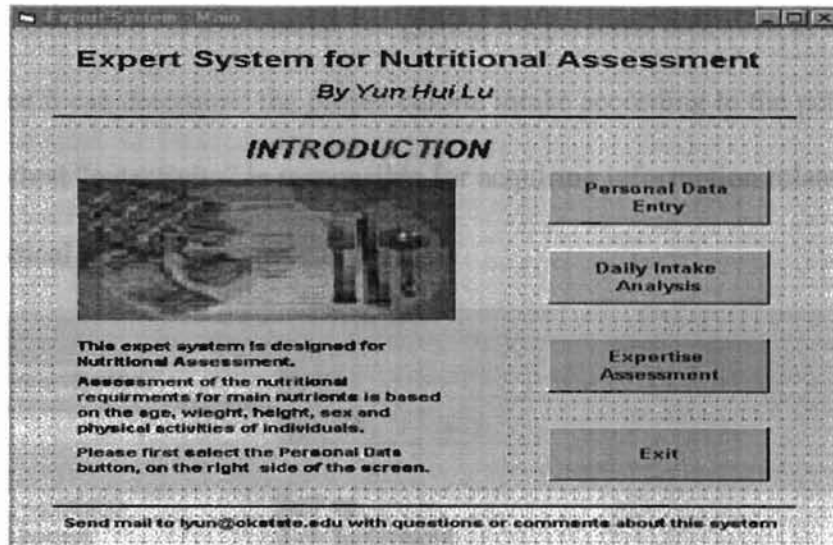
There are some other advantages to using the DLL. DLL is a standalone object and need not be built; all of the functions may still be directly called from the application.

## **CHAPTER III**

### **Design and Implementation**

#### **Program Design**

This expert system application consists of three components, namely the graphic user interface, the system database, and the rule-based inference mechanism. The interface of the application has the greatest impact on the user's opinion. It should let users discover the various features of the application easily without instructions. A customized Visual Basic GUI of the expert system has been created to access the database, and it will pass the message between CLIPS and users. To use this example expert system, the users need merely to load the VB ".exe" program into their PC even without the CLIPS software. Fig. 5 illustrates the main interface designed for this expert system application. As the "brain" of the whole system, the inference mechanism searches the matched rules through the knowledge base containing all the rules so as to arrive at a decision. To facilitate the functions of CLIPS, the rules are written in a file with a ".clp" extension. In order to enlarge the nutrition knowledge base, the nutrition expert knowledge data is contained in the Access database file with a ".mdb" extension.



**Figure 5.** The Main Application Window

### Implementation of the GUI

Before construction of the rule-based inference engine, the first concern is to provide a graphic user interface by Visual Basic 5.0 for the execution of the entire application. The Windows-based Visual Basic is a comprehensive object-oriented programming tool. A number of VB built-in tools can be used to develop objects. Hence VB5 is chosen to develop the code for the GUI design of this expert system application.

In object-oriented programming, objects contain elements, such as variables, attributes, and methods. Each object is responsible for performing some duties depending on the methods and attributes encompassed within the object. In Figure 6, the right hand column indicates the object members (which includes attributes and methods) of the “personalinfo” object. The “flying box” icon indicates the methods (each method is responsible for a specific task), and the “pointing hand” icons indicate the attributes. For

example, “personalinfo” is responsible for getting the personal data such as sex and weight, so that it can determine the proper calorie intake according to the personal inputs. Also, the method “getactivity” is responsible for acquiring information related to the different physical activity level (See Fig. 7).

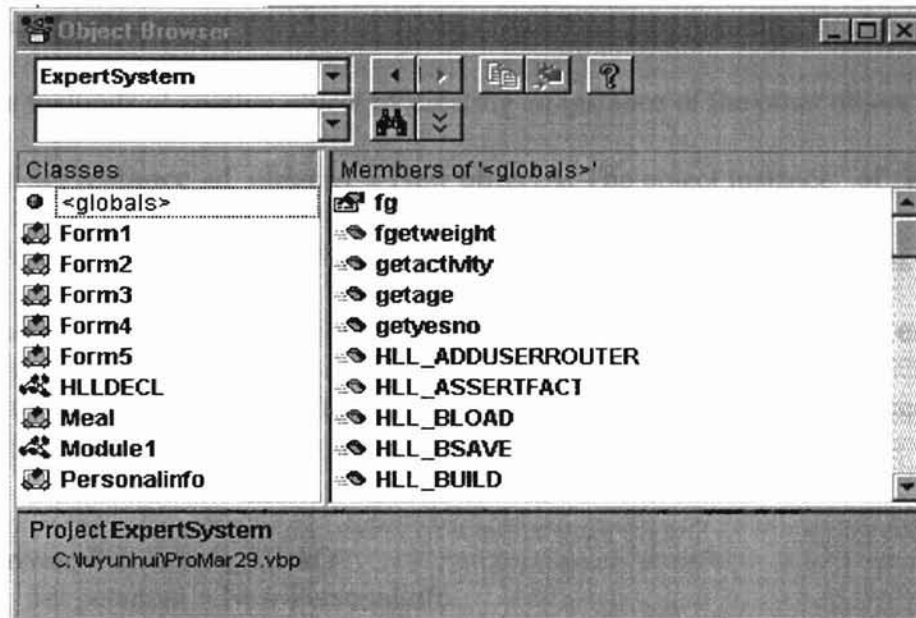


Figure 6. A List of Attributes and Methods with VB5

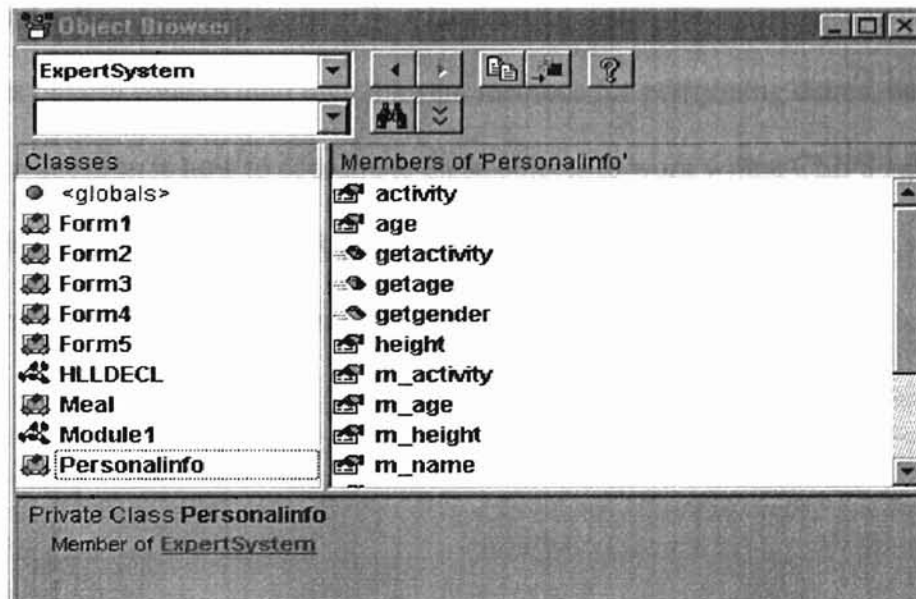


Figure 7. Attributes and Methods of Personalinfo Object

The pseudocode of the “getactivity” function is shown as follows:

```
Public Function getactivity() As String
    Dim actvt As String,
    Let actvt = m_activity,
    getactivity = actvt
End Function
```

Based on the features of the objects in object-oriented programming, one object can access the methods of another object by creating an instance of the other object using the command **Set instance\_of\_objectA = New objectA**. The object instance\_of\_objectA is now an instance of objectA, and it can access some methods of objectA as long as these methods are declared to be publicly accessible. Objects can communicate and exchange information by the virtue of this feature. The sample code written in VB5 is presented below:

```
Private Sub Result_Click()
    Set personal = New Personalinfo
    Let personal.activity = Combo1(0)
    Let at = personal.getactivity
    Text2.text = “Your choice for physical activity is ” & at
End sub
```

Various objects contain their own relevant methods for performing duties, but the next immediate question is how to coordinate these objects to work with a CLIPS ruled-based expert system to transfer the input data and retrieve the output data. This issue will be dealt with in the following section.

### Embedded CLIPS within VB5 by DLL

In order to let the user interact with the expert system easily, the actions of a rule deduced from the inference engine need to be carried out by the object-orient module. The objects in the VB5 module are able to carry out the questions and monitor the status of the rules fired from the rule-based inference engine. In order to ensure smooth and efficient exchange of information between the rule-based inference engine and VB5 module, it is important that they are working under the same operating environment. In this study, the rule-based inference engine is developed within the CLIPS shell, while the object-orient module is developed with the VB5 tool; these two development tools do not naturally “talk” to each other. In this respect, it is important that these two modules must be integrated so as to achieve efficient forward and backward information transfer.

For implementing the tasks, some DLL files are necessary. Fortunately, in Microsoft Windows, a Dynamic Link Library (DLL) can be easily developed to link Windows-based products to achieve information exchange among the software applications. As a matter of fact, a DLL for CLIPS 6.0 is also available.

For integrating CLIPS to VB5, a special module named HLLDECL is written. It is used to call two DLL files: the clips.dll (32 bit), and the interface DLL - clipshll.dll (32 bit). The clips.dll file contains all the common CLIPS functions and the clipshll.dll simply provides a handy way that let VB5 application to access the CLIPS functions. It integrates the WINAPI (Window Application Program Interface) with the routines in

CLIPS. It is achieved by those function declarations in the HLLDECL module. Note that some of the declarations are function declarations and others are subroutine declarations.

All parameters are called by value. One of the declare function is shown as follows:

```
Declare Function HLL_LOAD Lib "CLIPSHLL.DLL" (ByVal FileName As String, ByVal FileNameLen as integer) As Integer
```

Parameters:

FileName: String containing full filename/path of CLP file to load

FileNameLen: Length of the File Name. (i.e. For the file c:\rules.clp this parameter is 12)

Return values:

-2 :bad filename

-1 :string buffer containing filename is no valid

0 :load failed

1 :Parse Failed

2 :Load OK

3 :Bad Load name

4 :Init not called (should never happen in HLL\_)

Comments: load a CLP file from disk into CLIPS

For communication between VB5 modules and CLIPS program, CLIPS I/O routines must be named by the use of logical names. The use of logical names allows CLIPS to ignore the specifics of an I/O request. There are also several logical names predefined by CLIPS, such as "wdisplay", "stdout", "werror", "wwarning", "wdialog", "wagenda", and "wtrace". When a condition that requires user input is met, the program will examine the agenda conditions to see what new information is needed and then gather the answers from the user. Fig. 8 shows the user can input the data through the dialog interface. The program will then capture a specific router, and send the answer to the route. Then the related information is pushed back into the agenda through the route. This will cause more rules to be placed on the agenda and the new facts will be asserted as new values.

With these DLL applications added to VB5, the inference mechanism of CLIPS becomes a part of VB5, thus enabling free and automatic data exchange between these

two applications. Fig. 9 illustrates the interaction between the user and this expert system in the Windows-based GUI.

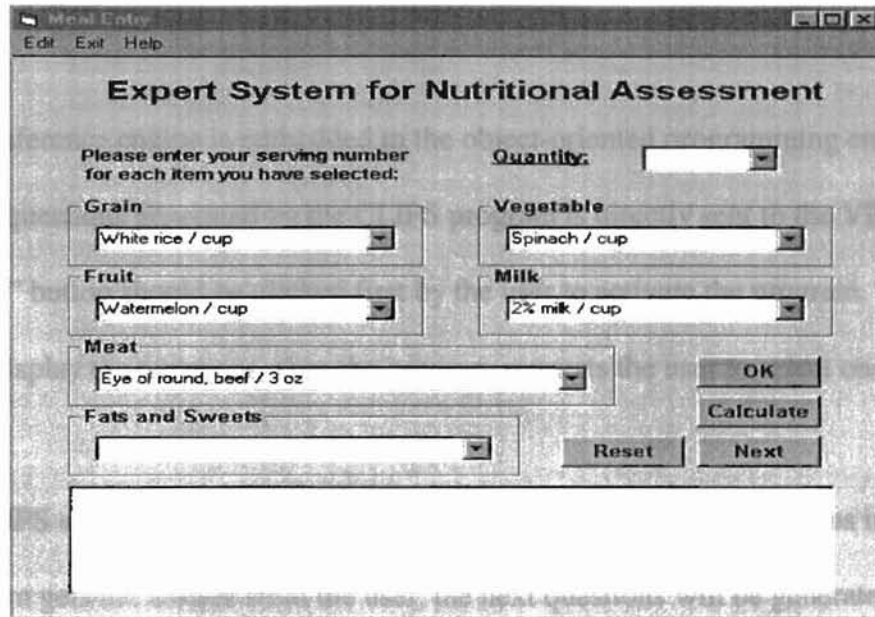


Figure 8. The Interface for the User to Input Actual Daily Intake

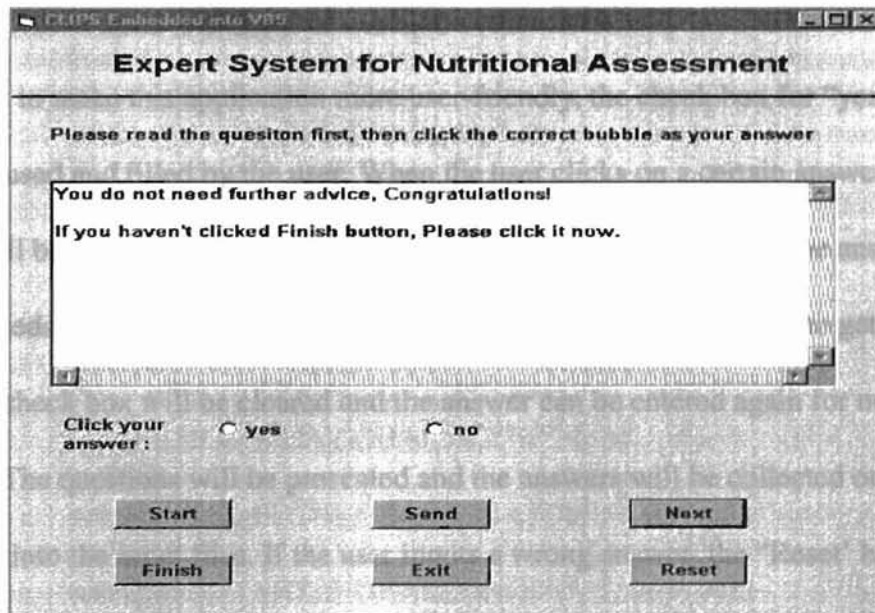


Figure 9. The Interface of the CLIPS Expert System Embedded into VB5



## Facilities of VB GUI

As the inference engine is embedded in the object-oriented programming environment, the list of questions generated by the CLIPS program is directly sent to the VB5 program. The “Start” button should be clicked first by the user to activate the program. The text box will display the questions, then the program requests the user to select one answer at a time.

In a CLIPS shell, there is no opportunity to go back and alter the previous input. After the program gets the answer from the user, the next questions will be generated according to the previous answer. If the user inputs a wrong answer, he or she needs to restart the program all over again with a set of execution commands.

In order to make this application more user-friendly, the check box for “yes” or “no” answer is used and filled by the user. When the user clicks on a certain answer, then the answer will be entered into the system. When the user is satisfied with the answer, he or she just needs to click the “Send” button followed by the “Next” button to get to the next step. The check box will be cleared and the answer can be entered again for next question. The questions will be presented and the answers will be collected one by one, and saved into the input files. If the user inputs a wrong answer, the “Reset” button is designed specially for the user to clear the wrong answer and start the questions again without restarting the program. Finally, a command button in VB5 called **Finish** is used

to mask a set of CLIPS program commands, such as load, reset and run. To execute the CLIPS program. The code in VB5 is shown below:

```
Private Sub Finish_Click()  
    Dim retval As Integer  
    Dim FileName As String  
    Dim Showresult As String  
    Dim textline As String  
    Dim Ret As String * 256  
    Dim retint As Integer  
    Dim retfloat As Single  
    Dim retlong As Long  
    Dim retstring As String * 256  
    Dim path As String, name As String  
    path = Dir1.path  
    name = path & "\result.txt"  
    FileName = path & "\Nutrition.CLP"  
    Close #1  
    Form5.Text2.Text = ""  
    retval = HLL_LOAD(FileName, Len(FileName))  
    If retval < 1 Then  
        MsgBox ("Can not Load " & FileName)  
        Exit Sub  
    End If  
    HLL_RESET  
    HLL_CLEARROUTE "wdisplay", 8  
    HLL_CLEARROUTE "stdout", 6  
    HLL_CLEARROUTE "werror", 6  
    HLL_CLEARROUTE "wdialog", 7  
    HLL_CLEARROUTE "wagenda", 7  
    HLL_CLEARROUTE "wtrace", 6  
    HLL_CLEARROUTE "wclips", 6  
    HLL_CLEARROUTE "wwarning", 8  
    retval = HLL_SETGLOBALINT("w", 1, 3)  
    retfloat = 3.3  
    retval = HLL_SETGLOBALFLOAT("x", 1, retfloat)  
    retfloat = 0  
    retval = HLL_SETGLOBALLONG("y", 1, 999999)  
    retval = HLL_SETGLOBALSTRING("z", 1, "asdasd", 6)  
    retval = HLL_RUN(-1)  
    If retval <= 0 Then  
        MsgBox "File doesn't not run."  
        Exit Sub  
    End If
```

```

        Showresult = "    ===== The Assessment from the Expert System =====" _
            & vbCrLf & vbCrLf
        Form5.Text2.Text = Showresult
        Open name For Input As #2
        Do While Not EOF(2)           ' Loop until end of file.
            Line Input #2, textline   ' Read line into variable.
            Showresult = Showresult & textline & vbCrLf
        Loop
        Close #2                     ' Close file.
        Form5.Text2.Text = Showresult
    End Sub

```

Every question would be in a yes-no format, or in a multiple-choice format. Any answer from the user would invoke the corresponding rules to fire. The function for the assignment of questions consists mainly of a *Select Case* statement. All the possible questions are included to carry out the task. The code of this function is shown below:

```

Private Sub Next_Click()
    Dim flag As Boolean
    flag = False
    Select Case (question)
    Case 0
        Text2.Text = "Do you know how to keep a healthy dietary habit
            for daily servings (yes/no)? "
        answer = getyesno()
        If answer = "yes" Then
            question = 1
        ElseIf answer = "no" Then
            question = 2
        End If    code continued ...
    End sub

```

As shown above, the *Select Case* block can be used as part of a decision-making structure because it helps to simplify the choosing among several questions. After the user has finished the whole program process, the user can click on the "Exit" button to exit the program. The user is in control in VB. Therefore, the CLIPS program has been

successfully embedded within VB application to form a fully integrated expert system for nutrition assessment.

### **Implementation of the Rule-based Expert System**

Rule-based programming is one of the most common techniques for developing expert systems. Typically, rules are used to represent heuristics in nature, based on useful “rules of thumb” which specify a set of actions that can be applied to a given situation. A rule-based expert system includes knowledge base, facts and data. The expert system knowledge base will typically be encoded as a set of IF-THEN rules that expresses how the facts can be evaluated. The case-specific data includes the input data provided by the user, the partial conclusions, and the output data produced by the program. In a simple forward chaining rule-based system, the case-specific data will be stored in working memory. The working memory is used to store transient and dynamic information data values such as the rules that have been fired upon the instantiation of the facts within, and so on.

A rule is composed of an IF portion on the LHS and a THEN portion on the RHS of a rule. The IF portion of a rule is a series of matching patterns. These patterns specify which fact would cause the rule to be applicable. The process of matching facts to patterns is called “pattern matching”. The Expert system provides a mechanism, called the *inference engine*, which automatically matches facts against patterns, and determines which rules are applicable. The IF portion of a rule can actually be thought of as the “whenever” portion of a rule since a pattern matching always occurs whenever facts list

are changed. The THEN portion of a rule is the set of actions to be executed when the rule is applicable. The general format of a rule looks like below:

```
(defrule <rule name> [<optional comment>] ; Name of a rule
  <<patterns>> ; the left-hand Side (LHS) of the rule, or the IF portion
=>
  <<actions>>) ; the right-hand Side (LHS) of the rule, or the THEN
                portion
```

The actions of an applicable rule are executed by inference engine. When the inference engine is instructed to execute, it selects a rule, and then executes the actions of the selected rule. The inference engine will select another rule and executes its actions, until no applicable rules remain.

For example, a set of the possible inference processes in the nutrition expert system is shown below:

**IF:** Question: "Do you have the servings that are right for you " = "Yes"

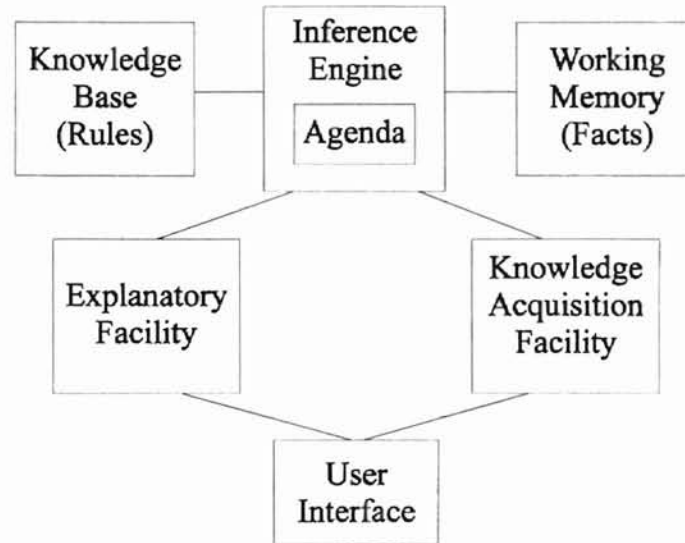
**Then:** Next rule = diet intake normal

**IF:** Question: "Do you have the servings that are right for you" = "No"

**Then:** Next rule = diet intake abnormal

In a rule-based system, the inference engine determines which rules are satisfied by the facts. Two general methods of inference are commonly used as the problem solving strategies of expert systems: forward chaining, and backward chaining. The forward chaining method is a data-driven process that is deduction from facts to the conclusions. The backward chaining method is a goal-driven process that involves reasoning in reverse from a hypothesis up to the facts that support the hypothesis. To establish and support this

hypothesis, a series of questions is asked. If the response is yes, then the hypothesis is proven true and would become a fact (See Fig. 10).

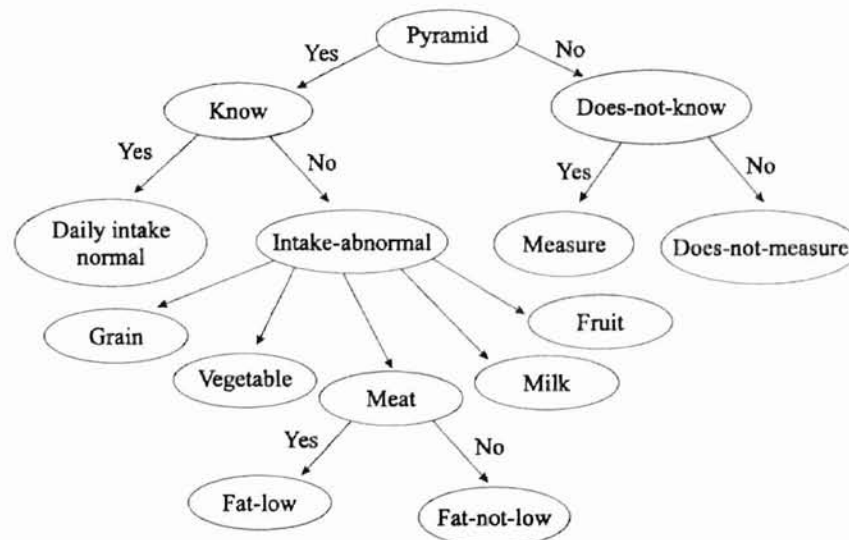


**Figure 10.** Structure of a Rule-Based Expert System

### **Implementation of the Inference Engine**

The inference engine is the core of the CLIPS expert system program. The inference mechanism, through its use of the knowledge base to arrive at the goal, is able to fire a series of rules that would provide for an educated diagnosis of the problems. To implement the inference engine, a decision tree is used. Each decision node represents a question or decision which, when answered or decided, determines the appropriate branch of the decision tree to follow. A decision structure is both a knowledge representation scheme and a method of reasoning. It can be used to construct all the rules in the inference structure. Every rule is represented by a node in the decision tree, with each node linked to one another by the branches, resulting in a cascading chain of rules fired in

the inference process. Rules are extracted from a decision tree by tracing the path from a leaf node to the root node of the tree. The dependent-attribute value assigned to this node becomes the right-hand side of a rule, and the independent attributes and values in one node on the path to the root become the conditions on the left-hand side to that rule. One useful feature of the decision tree is that it can be self-learning. During the query process of the user for the answers by the expert system, after the user has provided the first answer, the decision tree would use this answer from the user to select the next appropriate question. Then new nodes, branches and leaves can be created dynamically and added to the tree. In the expert system program, the inference process is initiated by setting the first rule to the root node of the decision tree (See Fig 11).



**Figure 11.** Nutrition Decision Tree with Multiple Branches

For example, if the question in the decision tree is “Do you know the Food Guide Pyramid?”, then the left branch of the node would represent the path to follow the answer “yes”, and the right branch of the node would represent the path to follow the answer “no”.

## Design of Facts and Templates

The templates have a well-defined structure. They are basically used in writing rules. In the process of the breakdown of a user's request, templates should be designed to suit the overall requirements, particularly taking into consideration the operations of the inference process. Groups of facts can be described by using a template. A fact in the CLIPS program consists of one or more fields enclosed in matching left and right parentheses, and facts can be represented in a meaningful manner such as (<relation name><value1> ... <valueN>). An example is the (fats low), (fats not-low) which explicitly declares the relationships between various values. Facts are normally asserted during the start of the inference process that operates with selecting the rules, matching the symbols of facts, and then "firing" the rules to establish new facts. The assertion of facts is analogous to the initializing of a structured program, where the variables are assigned with certain values. In this rule-based program, the facts make use of the templates, such as answer-value, activity, phase, f-age and m-weight are shown as below examples: (answer-value ?value), (activity get-level-f)(activity get-level-m), (phase read-data) (phase close-files), (f-age 50)(f-age30)(f-age20), (m-weight 200)(m-weight 150)(m-weight 120), and so on.

It should be noted that during the inference process, the attributes of the facts are changing continuously depending on which rules are fired. For example, the fact (phase read-data) in first rule "open-files" indicates that at the beginning of the program, the first rule without any patterns will be automatically assigned a pattern (initial-fact) to its LHS,



and thus the rule1 will be placed on the agenda when a command “reset” is performed. Then new facts (phase read-data) and (phase0 start) will be added to the fact-list. Another point that needs to be explained here is that only new facts that are entered will be seen by the rule. The rule does not fire otherwise.

### Formulation of Rules

In any rule-based expert system, rules are necessary to infer new facts from the data in the knowledge base, and the facts are asserted and modified during the inference process. In most cases and also in this program, the assertion of the facts starts at the time when the first rule is fired. The pseudo-code for one of the possible rules in this expert system is shown below:

**IF** you have the number of servings from the food groups that are right for you  
**THEN** your daily intake is normal

Shown following is the rule expressed in CLIPS syntax:

```
(defrule know-FGP-yes
  ?phase5 <- (phase5 know-FGP)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase5 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Do you have the right number of servings from the
                    groups(yes/no)?" crlf)
  (assert (phase6 intake-normal))
  (assert (phase read-data)))
```

The CLIPS 6.10 main menu allows the user using the “load” command to load the rules. The CLIPS program will not start running unless there are rules whose LHS are

satisfied by some facts. The “reset” command asserts a default startup fact that is generated by reset. This fact is called as initial-fact. Also the “reset” command removes all previous activated rules from the agenda and all old facts from the fact-list. When a CLIPS program is run by using the “run” command, the rule with the highest salience in the agenda is fired first. Fig. 12 illustrates the activation of the rules in the working space.

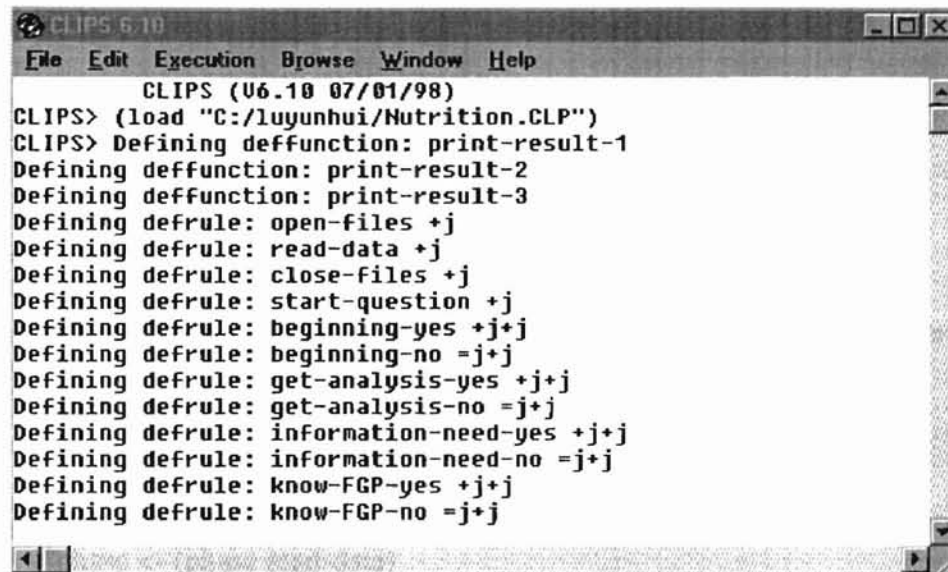


Figure 12. Rules in Agenda with CLIPS

For providing a clear picture of the inference procedures of the rules, a description of the start processing in execution of the rule-based program is included in the following context.

The startup rule of this program contains the assertion of the “read-data” and “start” facts. This action then would invoke the next rule “start-question”, and would invoke the “read-data” rule to read the data from an opened file. The code of this startup rule is as follows:

```
(defrule open-files
=>
  (open "yesno.dat" finput "r")
```

```
(open "result.txt" foutput "w")
(assert (phase read-data))
(assert (phase0 start)))
```

Because there is no pattern needed in the LHS of the “startup” rule, the statements of actions after “=>” are executed. As the assigned matching pattern of the LHS changes according to the input, the following rules satisfy the condition and therefore it is fired almost at the same time as the *first* firing rule and the *second* firing rule: (despite the *startup* rule)

```
(defrule start-question
  ?phase0 <- (phase0 start)
=>
  (retract ?phase0)
  (printout foutput "Do you know how to keep a healthy dietary habit for daily
                    servings(yes/no)?" crlf)
  (assert (phase1 begin))
  (assert (phase read-data)))
```

```
(defrule read-data
  ?phase <- (phase read-data)
=>
  (retract ?phase)
  (bind ?value (read finput))
  (if (eq ?value EOF)
      then (assert (phase close-files))
      else (assert (answer-value ?value))))
```

The two rules above are fired because the matching patterns on their LHS are satisfied. Therefore, the actions of their RHS are executed. First, the startup rule results in the action which is to start the question and change the action to “begin” yes-no answering. The second fired rule results in another action which is to obtain the value of input data from an opened file.

The third and fourth rule to be fired is due to the begin rule and are shown below:

```
(defrule beginning-yes
```

```

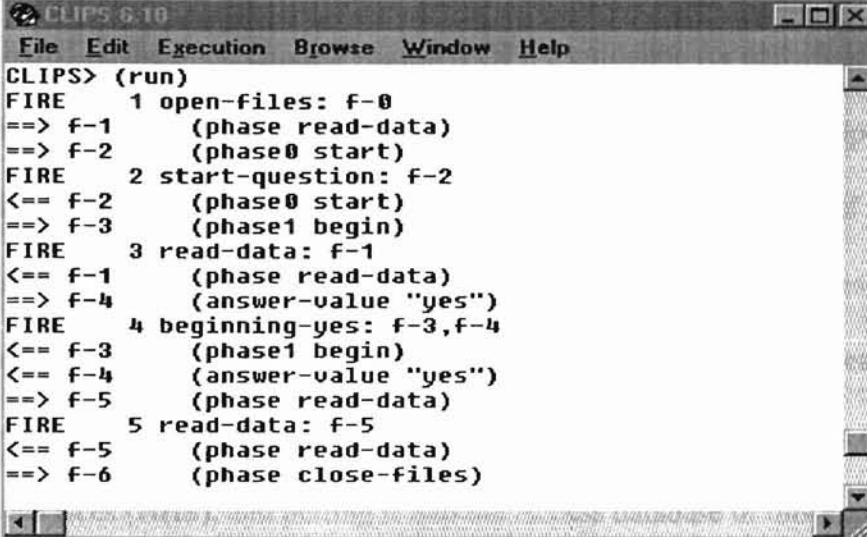
?phase1 <- (phase1 begin)
?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
(retract ?phase1 ?data)
(printout foutput "The answer is " ?yesno crlf crlf)
(printout foutput "You needn't to get further advice, Congratulation!" crlf)
(assert (phase read-data)))

(defrule beginning-no
  ?phase1 <- (phase1 begin)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
(retract ?phase1 ?data)
(printout foutput "The answer is " ?yesno crlf crlf)
(printout foutput "So, do you need me to give you an analysis that according to
your personal information (yes/no)?" crlf)
(assert (phase2 get-analysis))
(assert (phase read-data)))

```

Either of the previous two rules is fired due to the two statements on the LHS of the rules are satisfied. The “beginning-yes” rule will be fired only when a fact “begin” and an answer “yes” are asserted. In this case, the result is a new action which would provide a some advice to the user, and the read-data fact is asserted again. Notice that this is similar as the first fired rule, but the value of the variable “?value” is changed from “yes” to “EOF” since there is no more input data. Therefore the “read-data” rule will be fired again and assert the new facts “close-files”. In order to update a fact in the inference process, a “retract” is used. Every time, a new fact has been match to a rule, the facts should be retracted by the rule. The variables “?phase1” and “?data” used for storing facts are retracted when the “beginning-yes” rule is fired. It means that the facts stored in “?phase1” and “?data” are removed so that both variables can be reused to store new facts. If the answer “no” is asserted, and the two statements on the LHS of the “beginning-no” rule are satisfied, and the rule will be fired. The inference process is

manifested in the change of actions from one rule to another, and the chaining action will continue until the ultimate goal is satisfied. The recommended conclusion will then be deduced as a result of the inference mechanism. In this program, a total of 102 rules are used in the inference process (See Fig. 13).



```
CLIPS 6.10
File Edit Execution Browse Window Help
CLIPS> (run)
FIRE 1 open-files: f-0
==> f-1 (phase read-data)
==> f-2 (phase0 start)
FIRE 2 start-question: f-2
<== f-2 (phase0 start)
==> f-3 (phase1 begin)
FIRE 3 read-data: f-1
<== f-1 (phase read-data)
==> f-4 (answer-value "yes")
FIRE 4 beginning-yes: f-3,f-4
<== f-3 (phase1 begin)
<== f-4 (answer-value "yes")
==> f-5 (phase read-data)
FIRE 5 read-data: f-5
<== f-5 (phase read-data)
==> f-6 (phase close-files)
```

Figure 13. Watch Rules and Facts with CLIPS

In a CLIPS program, a fact is the basic method for controlling the execution of the inference process. Another control method in CLIPS is to provide a direct way through **salience**. Salience allows more important rules to stay at the top of the agenda regardless of when the rules were added to the agenda, so that the rules with a lower salience are pushed onto the agenda below the rules with a higher salience. The salience is set by assigning a numeric value to the rules, ranging from the smallest value of -10,000 to the highest of 10,000. If a rule has no salience explicitly assigned, CLIPS assumes its salience as 0. The salience is used in this program in order to ensure the “close-files” rule will be fired as the ultimate rule, since this rule will close all the input and output files.

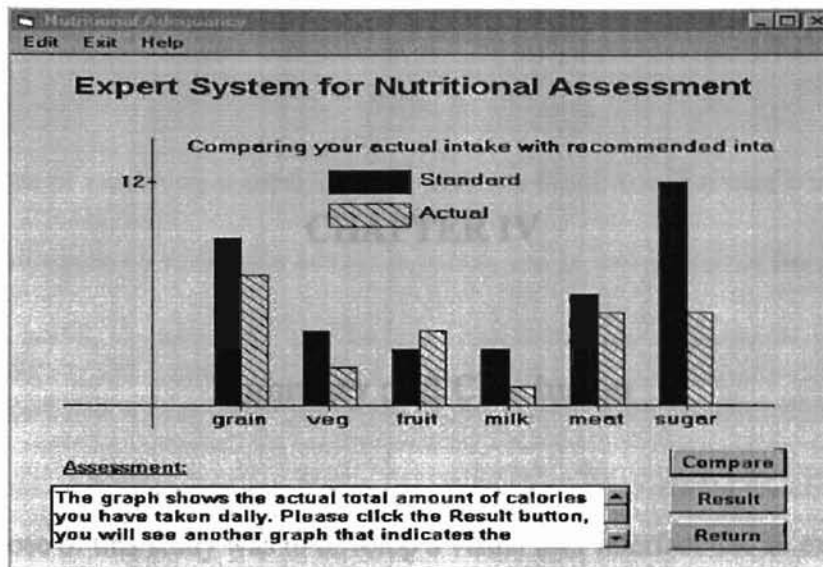
## **Database Supplement**

The basic nutrition information for this system is applied to a sample nutrition assessment based on the Food Guide Pyramid (FGP), a guideline for daily food choices developed by the United States Department of Agriculture (USDA). FGP helps users to put the dietary guidelines into action, so that the users can decide what kind of food to eat and how much of the food to eat to have all the nutrients needed without excess calories.

To set up the expert knowledge base of the program, this study first involves extracting the FGP servings data based on the statistics from USDA and the Department of Health and Human Services (HHS), and putting it into the Access database of this expert system.

The basic information for the standard nutrients in food is stored in a relational database file which is developed by Microsoft Access 7. The application then can use this database to do some caloric intake calculation for users. Normally, to an expert system, the larger its knowledge base is, the more efficient its problem solving ability will be. But to a CLIPS program for a large knowledge base, it may require more memory space and overhead. The use of the Access database enhanced this expert system application without more memory space and overhead.

Six tables refer to the six types of food groups in this database. Each table lists some nutrient values for specific, real-life servings of many different foods and beverages. Each entry in the table is a snapshot of a specific food serving that shows how much the serving weighs and how great the calorie counts.



**Figure 14.** Comparison of the Result of Food Servings that the User Provided

Fig. 14 illustrates a sample that displays the result of the comparison between the recommended and the actual caloric intake of the individual in a particular day. This comparison is based on the information in the database. In order to access the database, some control objects are placed on the several forms in the VB5 program. Therefore the nutritional information database is bound to a data control in VB5 by setting the DatabaseName to the specific database file and RecordSource properties of the objects to the field of the database. Then the program will calculate and analyze the caloric intake according to the user's input and expert knowledge in the database.

## CHAPTER IV

### Summary and Conclusion

The purpose of this study was to develop a visual user interface for an expert system. This was done by the integration of programs developed by different tools, such as CLIPS and Visual Basic. The example expert system used to illustrate this thesis was an expert system for nutrition assessment that can be usefully deployed in a nutrition consultation and education field.

It comprises a rule-based inference mechanism by using the CLIPS shell as the development tool, and a VB GUI program developed by the Visual Basic 5.0, as an object-oriented programming tool. The two development tools CLIPS and VB5 do not naturally “talk” to each other, so the Dynamic Link Library and Active Data Objects are used to integrate CLIPS with VB5 successfully so as to achieve an efficient forward and backward data transfer between the two programs. Also a relational database is developed using Microsoft Access as a component of the application.

By approaching the queries designed for this application, this expert system can help users to access some expert knowledge such as the Food Guidelines. Also it enables the users to seek some expert nutrition advice easier when the human experts are expensive, unavailable, overburdened, or scarce. It would therefore benefit both the experts and



users. Also, this expert system can be used simultaneously in many sites, and can provide consistent and uniform assessment and advice. In addition, this expert system is able to explain the line of reasoning it used for each problem it solved. The user will then be able to study the rationale on which the advice is based, and is free to accept the advice or reject it. As a result, the user will “get the best from their food”. Moreover, it is necessary that novice users can access the required information without the knowledge of using the CLIPS program and remembering all the textual commands to access a CLIPS expert system.

This paper provides an insight into the development of the CLIPS program and VB GUI, and the technology of integrating the CLIPS program and the VB program to form a Windows-based product. The main technique used is the DLL that lets the application achieve an efficient information exchange between different software components.

## BIBLIOGRAPHY

- [Barnes 1988] Linda Woodring Barnes. "Expert System for Computer Assisted Floristic Classification", M. S. Thesis, Computer Science, Oklahoma State University, 1988.
- [Bhargava 1993] Dipti R. Bhargava. "Building An Expert Database System in C Using CLIPS and Paradox", M. S. Thesis, Computer Science, Oklahoma State University, 1993.
- [Carrico, Girard and Jones 1989] Michael A. Carrico, John E. Girard and Jennifer P. Jones. *Building Knowledge Systems*, New York: McGraw-Hill Book Companies, Inc., 1989.
- [CNPP 1999] Center for Nutrition Policy and Promotion, U.S. Department of Agriculture, *The Food Guide Pyramid*, 1999.  
URL: [http://www.pueblo.gsa.gov/cic\\_text/food/food-pyramid/main.htm](http://www.pueblo.gsa.gov/cic_text/food/food-pyramid/main.htm)
- [Drummond 1997] Karen Eich Drummond. *Nutrition for the Foodservice Professional*, New York: Van Nostrand Reinhold, 1997.
- [DuPuy 1995] Nancy A. DuPuy. *Focus on Nutrition*, St. Louis: Mosby-Year Book, Inc., 1995.
- [Fisher and Rinzler 1997] Lynn Fischer and Carol Ann Rinzler. *Healthy Eating On-the-go for Dummies*, Foster City: IDG Books Worldwide, Inc., 1997.
- [FNIC 1999] The Food and Nutrition Information Center (FNIC), "Database of Food and Nutrition Software and Multimedia Programs", 1999.  
URL: <http://www.nal.usda.gov/fnic/software/software.html>
- [Giarratano and Riley 1989] Joseph Giarratano and Gary Riley. *Expert Systems*, Boston: PWS-KENT Publishing Company, 1989.
- [Hayes-Roth, Waterman and Lenat 1983] Frederick Hayes-Roth, Donald A. Waterman and Douglas B. Lenat. *Building Expert Systems*, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983.

- [Koh 1992] Swee-Hyong Koh. "An Advisory Expert System on Computer Procurement", M. S. Thesis, Computer Science, Oklahoma State University, 1992.
- [Kolasa 1997] Kathryn M. Kolasa. "New developments in nutrition education utilizing computer technology", *Nutrition Education for the Public*, (62), pp.179-209, 1997.
- [Lau, Tso and Ho 1998] H. C. W. Lau, S. K. Tso and J. K. L. Ho. "Development of an intelligent task management system in a manufacturing information network", *Expert Systems with Applications*, 15 (2), pp.165-179, 1998.
- [Lee and Nieman 1996] Robert D. Lee and David C. Nieman. *Nutritional Assessment*. St. Louis: Mosby-Year Book, Inc., 1996.
- [Leigh and Doherty 1986] William E. Leigh and Michael E. Doherty. *Decision Support and Expert Systems*, Cincinnati: South-Western Publishing Co., 1986.
- [Li 1999] Lynn Ling X Li. "Knowledge-based problem solving: an approach to health assessment", *Expert Systems with Applications*, 16 (1), pp.33-42, 1999.
- [Piper 1996] Brenda Piper. *Diet and Nutrition*, San Diego: Chapman & Hall, 1996.
- [Riley 2000] Gary Riley. URL: <http://www.ghgcorp.com/clips/download>
- [Schneider 1998] David I. Schneider. *An Introduction to Programming Using Visual Basic 5.0*, Upper Saddle River: Prentice Hall, Inc., 1998.
- [Silberschatz, Korth and Sudarshan 1999] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*, Boston: McGraw-Hill Book Companies, Inc., 1999.
- [Wardlaw and Insel 1996] Gordon M. Wardlaw, Paul M. Insel. *Perspective in Nutrition*, St. Louis: Mosby-Year Book, Inc., 1996.
- [Waterman 1986] Donald A. Waterman. *A Guide to Expert Systems*, Massachusetts: Addison-Wesley Publishing Company, Inc., 1986.
- [Wolke 1997] George Eric Wolke. "A Rule Based Expert System Which Configures Gas Chromatographs", M. S. Thesis, Computer Science, Oklahoma State University, 1997.

## APPENDIX

```
=====
;
;           Building an Expert System by
;           Integrating CLIPS with Visual Basic
;
;           Yun Hui Lu
;
;           Advisor: Jacques LaFrance
;
;           Computer Science Department
;           Oklahoma State University
;
;           May 31, 2000
;
;           This is the part of CLIPS code to implement
;           the inference engine for this expert system.
;=====

;=====
;deffunctions
;=====

(deffunction print-result-1 (?n)
  (printout foutput "For your caloric level, every day you should have
the following servings: " crlf)
  (printout foutput "6 servings for grain group;" crlf)
  (printout foutput "3 servings for vegetable group;" crlf)
  (printout foutput "2 servings for fruit group;" crlf)
  (printout foutput "2 servings for milk group;" crlf)
  (printout foutput "5 ounces for meat group;" crlf)
  (printout foutput "53 grams for total fats;" crlf)
  (printout foutput "6 teaspoons for total added sugars. " crlf))

(deffunction print-result-2 (?n)
  (printout foutput "For your caloric level, every day you should have
the following servings: " crlf)
  (printout foutput "9 servings for grain group;" crlf)
  (printout foutput "4 servings for vegetable group;" crlf)
  (printout foutput "3 servings for fruit group;" crlf)
  (printout foutput "3 servings for milk group;" crlf)
  (printout foutput "6 ounces for meat group;" crlf)
  (printout foutput "73 grams for total fats;" crlf)
  (printout foutput "12 teaspoons for total added sugars. " crlf))

(deffunction print-result-3 (?n)
  (printout foutput "For your caloric level, every day you should have
the following servings: " crlf)
```

```

(printout foutput "11 servings for grain group;" crlf)
(printout foutput "5 servings for vegetable group;" crlf)
(printout foutput "4 servings for fruit group;" crlf)
(printout foutput "3 servings for milk group;" crlf)
(printout foutput "7 ounces for meat group;" crlf)
(printout foutput "93 grams for total fats;" crlf)
(printout foutput "18 teaspoons for total added sugars. " crlf))

;=====
;rule0 open-files
;=====

(defrule open-files
=>
  (open "yesno.dat" finput "r")
  (open "result.txt" foutput "w")
  (assert (phase read-data))
  (assert (phase0 start)))

;=====
;rule1 read-data
;=====

(defrule read-data
  ?phase <- (phase read-data)
  =>
  (retract ?phase)
  (bind ?value (read finput))
  (if (eq ?value EOF)
      then (assert (phase close-files))
      else (assert (answer-value ?value))))

;=====
;rule2 close-files
;=====

(defrule close-files
  (declare (salience -99))
  ?phase<- (phase close-files)
  =>
  (retract ?phase)
  (assert (phase100 endrule))
  (close finput))

;=====
;rule3 start-question
;=====

(defrule start-question
  ?phase0 <- (phase0 start)
  =>
  (retract ?phase0)
  (printout foutput "Do you know how to keep a healthy dietary habit
for daily servings (yes/no)?" crlf)
  (assert (phase1 begin))
  (assert (phase read-data)))

```

```

;=====
;rule4 beginning-yes
;=====

(defrule beginning-yes
  ?phase1 <- (phase1 begin)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase1 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "You needn't get further advice, Congratulations!"
crlf)
  (assert (phase read-data)))

;=====
;rule5 beginning-no
;=====

(defrule beginning-no
  ?phase1 <- (phase1 begin)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase1 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "So, do you need me to give you an analysis
according to your personal information (yes/no)?" crlf)
  (assert (phase2 get-analysis))
  (assert (phase read-data)))

;=====
;rule6 get-analysis-yes
;=====

(defrule get-analysis-yes
  ?phase2 <- (phase2 get-analysis)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase2 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Since you do not know the right number of serving
for you," crlf)
  (printout foutput "This test needs you to answer the following
question, if you are a female," crlf)
  (printout foutput "please click yes, otherwise click no." crlf)

  (assert (phase3 gender-need))
  (assert (phase read-data)))

;=====
;rule7 get-analysis-no
;=====

(defrule get-analysis-no
  ?phase2 <- (phase2 get-analysis)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase2 ?data)

```

```

(printout foutput "The answer is " ?yesno crlf crlf)
(printout foutput "Well, do you need me to provide you some other
nutrition information(yes/no)?" crlf)
(assert (phase4 information-need))
(assert (phase read-data)))

;=====
;rule8 information-need-yes
;=====

(defrule information-need-yes
  ?phase4 <- (phase4 information-need)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase4 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Do you know the Food Guide Pyramid with six major
food groups (yes/no)?" crlf)
  (assert (phase5 know-FGP))
  (assert (phase read-data)))

;=====
;rule9 information-need-no
;=====

(defrule information-need-no
  ?phase4 <- (phase4 information-need)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase4 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "If you need to know more later, you are welcome to
use this expert system again." crlf)
  (assert (phase read-data)))

;=====
;rule10 know-FGP-yes
;=====

(defrule know-FGP-yes
  ?phase5 <- (phase5 know-FGP)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase5 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Do you have the right number of servings from the
groups (yes/no)?" crlf)
  (assert (phase6 intake-normal))
  (assert (phase read-data)))

;=====
;rule11 know-FGP-no
;=====

(defrule know-FGP-no
  ?phase5 <- (phase5 know-FGP)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))

```

```

=>
  (retract ?phase5 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "I will give you some guidelines about FGP." crlf)
  (printout foutput "The Food Guide Pyramid is made up of six food
groups, and" crlf)
  (printout foutput "the placement of these groups corresponds to the
recommended" crlf)
  (printout foutput "number of daily servings. The base of the pyramid
is made up" crlf)
  (printout foutput "of the largest group - breads, cereals, rice, and
pasta, and" crlf)
  (printout foutput "you should strive to eat the most servings from
this particular" crlf)
  (printout foutput "group. The tip of the triangle is formed by the
smallest group -" crlf)
  (printout foutput "fats, oils, and sweets, which you should consume
sparingly." crlf)
  (printout foutput "If you follow the pyramid and make wise choices,
you will most" crlf)
  (printout foutput "likely follow a lowfat eating plan." crlf)
  (assert (phase read-data)))

;=====
;rule12 intake-normal-yes
;=====

(defrule intake-normal-yes
  ?phase6 <- (phase6 intake-normal)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase6 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good, your daily intake is nutritious." crlf)
  (assert (phase read-data)))

;=====
;rule13 intake-normal-no
;=====

(defrule intake-normal-no
  ?phase6 <- (phase6 intake-normal)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase6 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "If you want to know more information about your
daily intake," crlf)
  (printout foutput "Please click yes, otherwise click no." crlf)
  (assert (phase7 more-details))
  (assert (phase read-data)))

;=====
;rule14 more-details-yes
;=====

(defrule more-details-yes

```



```

?phase7 <- (phase7 more-details)
?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase7 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Do your know how to measure your intake to get
good nutrition (yes/no)?" crlf)
  (assert (phase8 measure-intake))
  (assert (phase read-data)))

;=====
;rule15 more-details-no
;=====

(defrule more-details-no
  ?phase7 <- (phase7 more-details)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase7 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Thank you for using this expert system!" crlf)
  (assert (phase read-data)))

;=====
;rule16 measure-intake-yes
;=====

(defrule measure-intake-yes
  ?phase8 <- (phase8 measure-intake)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase8 ?data)
  (printout foutput "OK, you've got some knowledge already, the
recommended number of daily servings for you is: " crlf)
  (printout foutput "Grain group      --- 6-11 servings" crlf)
  (printout foutput "Vegetalbe group --- 3-5  servings" crlf)
  (printout foutput "Fruit group      --- 2-4  servings" crlf)
  (printout foutput "Meat group       --- 2-3  servings" crlf)
  (printout foutput "Milk group       --- 2-3  servings" crlf)
  (printout foutput "Fat & Sweets    --- use sparingly" crlf)
  (printout foutput "No more information, please click finish button."
crlf)
  (assert (phase read-data)))

;=====
;rule17 measure-intake-no
;=====

(defrule measure-intake-no
  ?phase8 <- (phase8 measure-intake)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase8 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Do you get at least 6 servings per day of Grain
(yes/no)?" crlf)
  (assert (phase9 get-grain))

```

```

(assert (phase read-data)))

;=====
;rule18 get-grain-yes
;=====

(defrule get-grain-yes
  ?phase9 <- (phase9 get-grain)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase9 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! You have got carbohydrates for your energy
from Grain." crlf)
  (printout foutput "Do you get at least 3 servings of vegetable daily
(yes/no)?" crlf)
  (assert (phase10 get-vegetable))
  (assert (phase read-data)))

;=====
;rule19 get-grain-no
;=====

(defrule get-grain-no
  ?phase9 <- (phase9 get-grain)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase9 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please get at least 6 servings of grain daily."
crlf)
  (assert (phase read-data)))

;=====
;rule20 get-vegetable-yes
;=====

(defrule get-vegetable-yes
  ?phase10 <- (phase10 get-vegetable)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase10 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! You have got Vitamins, minerals and Fiber
from vegetable" crlf)
  (printout foutput "Do you get at least 2 servings of fruit daily
(yes/no)?" crlf)
  (assert (phase11 get-fruit))
  (assert (phase read-data)))

;=====
;rule21 get-vegetable-no
;=====

(defrule get-vegetable-no

```

```

    ?phase10 <- (phase10 get-vegetable)
    ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
    (retract ?phase10 ?data)
    (printout foutput "The answer is " ?yesno crlf crlf)
    (printout foutput "Please get at least 3 servings of vegetable
daily." crlf)
    (assert (phase read-data)))

;=====
;rule22 get-fruit-yes
;=====

(defrule get-fruit-yes
  ?phase11 <- (phase11 get-fruit)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase11 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! You have got Vitamins, minerals and Fiber
from fruit." crlf)
  (printout foutput "Do you get at least 2 servings of meat daily
(yes/no)?" crlf)
  (assert (phase12 get-meat))
  (assert (phase read-data)))

;=====
;rule23 get-fruit-no
;=====

(defrule get-fruit-no
  ?phase11 <- (phase11 get-fruit)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase11 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please get at least 2 servings of fruit daily."
crlf)
  (assert (phase read-data)))

;=====
;rule24 get-meat-yes
;=====

(defrule get-meat-yes
  ?phase12 <- (phase12 get-meat)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase12 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! You have got Protein, Vitamins and Calcium,
Iron and Zinc from meat." crlf)
  (printout foutput "Please choose lean meat, poultry without skin to
keep your total fat and saturated fat low." crlf)
  (printout foutput "Do you get at least 2 servings of Milk daily
(yes/no)?" crlf)
  (assert (phase13 get-milk))

```

```

(assert (phase read-data)))

;=====
;rule25 get-meat-no
;=====

(defrule get-meat-no
  ?phase12 <- (phase12 get-meat)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase12 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please get at least 2 servings of meat daily."
crlf)
  (assert (phase read-data)))

;=====
;rule26 get-milk-yes
;=====

(defrule get-milk-yes
  ?phase13 <- (phase13 get-milk)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase13 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! You've got protein, Vitamins and Calcium
from milk." crlf)
  (printout foutput "Do you eat less fats and sweets daily (yes/no)? "
crlf)
  (assert (phase14 get-fats-Sweets))
  (assert (phase read-data)))

;=====
;rule27 get-milk-no
;=====

(defrule get-milk-no
  ?phase13 <- (phase13 get-milk)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase13 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please get at least 2 servings of milk daily."
crlf)
  (assert (phase read-data)))

;=====
;rule28 get-fats-Sweets-yes
;=====

(defrule get-fats-Sweets-yes
  ?phase14 <- (phase14 get-fats-Sweets)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase14 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)

```

```

(printout foutput "Good! Please use fats, oils and sweets sparingly."
crlf)
(printout foutput "Do you know how many calories will be consuming on
average per day (yes/no)?" crlf)
(assert (phase15 calories-consume))
(assert (phase read-data)))

;=====
;rule29 get-fat-Sweets-no
;=====

(defrule get-fats-Sweets-no
  ?phase14 <- (phase14 get-fats-Sweets)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase14 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please balance to use it. Saturated fat and
cholesterol are closely connected with heart disease." crlf)
  (assert (phase read-data)))

;=====
;rule30 calories-consume-yes
;=====

(defrule calories-consume-yes
  ?phase15 <- (phase15 calories-consume)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase15 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Good! Now you can follow the pyramid and make wise
choices when you are eating." crlf)
  (printout foutput "The information is ended here, please click finish
button." crlf)
  (assert (phase read-data)))

;=====
;rule31 calories-consume-no
;=====

(defrule calories-consume-no
  ?phase15 <- (phase15 calories-consume)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase15 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "The average you will be consuming is 2000 calories
a day, so please intake based on that." crlf)
  (printout foutput "The information is ended here, please click the
finish button.." crlf)
  (assert (phase read-data)))

;=====
;rule32 gender-female
;=====

```

```

(defrule gender-female
  ?phase3 <- (phase3 gender-need)
  ?data <- (answer-value ?yesno&:(eq ?yesno "yes"))
=>
  (retract ?phase3 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please choose your weight in pounds (above
145/bewteen 145-120/below 120)." crlf)
  (assert (phases16 female-weight))
  (assert (phase read-data)))

;=====
;rule33 gender-male
;=====

(defrule gender-male
  ?phase3 <- (phase3 gender-need)
  ?data <- (answer-value ?yesno&:(eq ?yesno "no"))
=>
  (retract ?phase3 ?data)
  (printout foutput "The answer is " ?yesno crlf crlf)
  (printout foutput "Please choose your weight in pounds (above
170/bewteen 170-145/below 145)." crlf)
  (assert (phases17 male-weight))
  (assert (phase read-data)))

;=====
;rule34 female-weight1
;=====

(defrule female-weight1
  ?phases16 <- (phases16 female-weight)
  ?data <- (answer-value ?yesno&:(eq ?yesno "145"))
=>
  (retract ?phases16 ?data)
  (printout foutput "The answer is above 145." crlf crlf)
  (printout foutput "Please choose your age range (above 50/bewteen 50-
20/below 20)." crlf)
  (assert (f-weight 145))
  (assert (phases18 get-age-female))
  (assert (phase read-data)))

;=====
;rule35 female-weight2
;=====

(defrule female-weight2
  ?phases16 <- (phases16 female-weight)
  ?data <- (answer-value ?yesno&:(eq ?yesno "130"))
=>
  (retract ?phases16 ?data)
  (printout foutput "The answer is between 145 and 120." crlf crlf)
  (printout foutput "Please choose your age range (above 50/bewteen 50-
20/below 20)." crlf)
  (assert (f-weight 130))
  (assert (phases18 get-age-female))

```

```

(assert (phase read-data)))

;=====
;rule36 female-weight3
;=====

(defrule female-weight3
  ?phase16 <- (phase16 female-weight)
  ?data <- (answer-value ?yesno&:(eq ?yesno "110"))
=>
  (retract ?phase16 ?data)
  (printout foutput "The answer is below 120." crlf crlf)
  (printout foutput "Please choose your age range (above 50/between 50-
20/below 20)." crlf)
  (assert (f-weight 110))
  (assert (phase18 get-age-female))
  (assert (phase read-data)))

;=====
;rule37 get-age-female-50
;=====

(defrule get-age-female-50
  ?phase18 <- (phase18 get-age-female)
  ?data <- (answer-value ?yesno&:(eq ?yesno "50"))
=>
  (retract ?phase18 ?data)
  (printout foutput "The answer is above 50." crlf crlf)
  (assert (f-age 50))
  (printout foutput "Please choose your activity
level(light/moderate/heavy)." crlf)
  (assert (activity get-level-f))
  (assert (phase read-data)))

;=====
;rule38 get-age-female-30
;=====

(defrule get-age-female-30
  ?phase18 <- (phase18 get-age-female)
  ?data <- (answer-value ?yesno&:(eq ?yesno "30"))
=>
  (retract ?phase18 ?data)
  (printout foutput "The answer is between 50 and 20." crlf crlf)
  (assert (f-age 30))
  (printout foutput "Please choose your activity
level(light/moderate/heavy)." crlf)
  (assert (activity get-level-f))
  (assert (phase read-data)))

;=====
;rule39 get-age-female-20
;=====

```

```

(defrule get-age-female-20
  ?phase18 <- (phase18 get-age-female)
  ?data <- (answer-value ?yesno&:(eq ?yesno "20"))
=>
  (retract ?phase18 ?data)
  (printout foutput "The answer is below 20." crlf crlf)
  (assert (f-age 20))
  (printout foutput "Please choose your activity
level(light/moderate/heavy)." crlf)
  (assert (activity get-level-f))
  (assert (phase read-data)))

;=====
;rule46 activity-level-f-light-145-50
;=====

(defrule f-light-145-50
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 145) (eq ?age 50)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-2 "1")
  (assert (phase read-data)))

;=====
;rule47 activity-level-f-light-145-30
;=====

(defrule f-light-145-30
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 145) (eq ?age 30)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule48 activity-level-f-light-145-20
;=====

(defrule f-light-145-20
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 145) (eq ?age 20)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)

```



```

(print-result-2 "1")
(assert (phase read-data)))

;=====
;rule49 activity-level-f-light-130-50
;=====

(defrule f-light-130-50
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 130) (eq ?age 50)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule50 activity-level-f-light-130-30
;=====

(defrule f-light-130-30
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 130) (eq ?age 30)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule51 activity-level-f-light-130-20
;=====

(defrule f-light-130-20
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 130) (eq ?age 20)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-2 "1")
  (assert (phase read-data)))

;=====
;rule52 activity-level-f-light-110-50
;=====

```

```

(defrule f-light-110-50
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 110) (eq ?age 50)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule53 activity-level-f-light-110-30
;=====

(defrule f-light-110-30
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 110) (eq ?age 30)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule54 activity-level-f-light-110-20
;=====

(defrule f-light-110-20
  ?activity <- (activity get-level-f)
  ?data <- (answer-value ?yesno&:(eq ?yesno "light"))
  (f-weight ?weight)
  (f-age ?age)
  (test (and (eq ?weight 110) (eq ?age 20)))
=>
  (retract ?activity ?data)
  (printout foutput "The answer is light." crlf crlf)
  (print-result-1 "1")
  (assert (phase read-data)))

;=====
;rule100 end-rule
;=====

(defrule end-rule
  (declare (salience -100))
  ?phase100 <- (phase100 end-rule)
=>
  (retract ?phase100)
  (printout foutput crlf crlf)
  (printout foutput "The expert system is finished." crlf crlf)
  (assert (phase101 close-output)))

```

```
;=====
;rule101 close-output
;=====
(defrule close-output
  (declare (saliency -1000))
  ?phase101 <- (phase101 close-output)
=>
  (retract ?phase101)
  (close foutput))
```

```

'=====
'
' The Part of Visual Basic code implemented as object-oriented module
' for achieving user interaction with CLIPS expert system.
'
'=====

```

```
Form3 (MealEntry.frm)
```

```
Option Explicit
```

```

Dim unitarray(1 To 6) As Variant
Dim qtarray(1 To 6, 1 To 5) As Single
Dim foodarray(1 To 6, 1 To 5) As String
Dim flag(1 To 6) As Boolean
Dim i As Integer
Dim j As Integer
Dim re As Integer
Private getentry As Meal
Const gr = 1, ve = 2, fr = 6, mi = 3, mea = 4, tip = 5

```

```

Private Sub combnum_Click()
    qtarray(i, j) = Val(combnum)

```

```
End Sub
```

```
Private Sub Comb01_Click()
```

```

    foodarray(i, j) = Comb01
    combnum.Text = ""

```

```
End Sub
```

```

Private Sub Comb01_dropdown()
    Static count As Integer

```

```

    unit.Clear
    Me.unit.AddItem "bird", 0
    Me.unit.AddItem "cup", 1
    Me.unit.AddItem "1/2 chicken", 2
    Me.unit.AddItem "large", 3
    Me.unit.AddItem "4 large", 4
    Me.unit.AddItem "leg", 5
    Me.unit.AddItem "6 medium", 6
    Me.unit.AddItem "3 oz", 7
    Me.unit.AddItem "20 small calms", 8
    Me.unit.AddItem "slice", 9

```

```

    If flag(mea) = False Then
        count = 0
        flag(mea) = True
    End If

```

```

    count = count + 1
    If count > 5 Then
        re = MsgBox("You can not select more than 5 items.", vbOKOnly)
        Exit Sub
    End If

```

```

End If
i = mea
j = count

Text1.Text = ""

End Sub

Private Sub Combo3_Click()
    foodarray(i, j) = Combo3
    combnum.Text = ""
End Sub

Private Sub Combo3_dropdown()
    Static count As Integer
    Me.unit.Clear

    Me.unit.AddItem "bagel", 0
    Me.unit.AddItem "biscuit", 1
    Me.unit.AddItem "cup", 2
    Me.unit.AddItem "euphrates", 3
    Me.unit.AddItem "large", 4
    Me.unit.AddItem "large slice", 5
    Me.unit.AddItem "medium", 6
    Me.unit.AddItem "muffin", 7
    Me.unit.AddItem "roll", 1
    Me.unit.AddItem "slice", 0
    Me.unit.AddItem "small stick", 5

    If flag(gr) = False Then
        count = 0
        flag(gr) = True
    End If

    count = count + 1
    If count > 5 Then
        re = MsgBox("You can not select more than 5 items.", vbOKOnly)
        Exit Sub
    End If
    i = gr
    j = count

    Text1.Text = ""

End Sub

Private Sub Combo4_Click()
    foodarray(i, j) = Combo4
    combnum.Text = ""
End Sub

Private Sub Combo4_dropdown()
    Static count As Integer
    unit.Clear

    Me.unit.AddItem "cup", 0
    Me.unit.AddItem "fruit", 1

```

```

Me.unit.AddItem "large", 2
Me.unit.AddItem "medium", 3

If flag(fr) = False Then
    count = 0
    flag(fr) = True
End If

count = count + 1
If count > 5 Then
    re = MsgBox("You can not select more than 5 items.", vbOKOnly)
    Exit Sub
End If
i = fr
j = count

Text1.Text = ""

End Sub

Private Sub Combo5_Click()
    foodarray(i, j) = Combo5
    combnum.Text = ""
End Sub

Private Sub Combo5_dropdown()
    Static count
    unit.Clear

    Me.unit.AddItem "1/2 cup", 0
    Me.unit.AddItem "cup", 1
    Me.unit.AddItem "oz", 2
    Me.unit.AddItem "Tbsp", 3

    If flag(mi) = False Then
        count = 0
        flag(mi) = True
    End If

    count = count + 1
    If count > 5 Then
        re = MsgBox("You can not select more than 5 items.", vbOKOnly)
        Exit Sub
    End If
    i = mi
    j = count

    Text1.Text = ""
End Sub

Private Sub Combo6_Click()
    foodarray(i, j) = Combo6
    combnum.Text = ""
End Sub

Private Sub Combo6_dropdown()
    Static count As Integer

```

```

unit.Clear

Me.unit.AddItem "bottle", 0
Me.unit.AddItem "can", 1
Me.unit.AddItem "cup", 2
Me.unit.AddItem "fl oz", 3
Me.unit.AddItem "glass", 4
Me.unit.AddItem "oz", 5
Me.unit.AddItem "packet", 6
Me.unit.AddItem "Tbsp", 7
Me.unit.AddItem "tsp", 8

If flag(tip) = False Then
    count = 0
    flag(tip) = True
End If

count = count + 1
If count > 5 Then
    re = MsgBox("You can not select more than 5 items.", vbOKOnly)
    Exit Sub
End If
i = tip
j = count

Text1.Text = ""
End Sub

Private Sub Combo7_Click()
    foodarray(i, j) = Combo7
    combnum.Text = ""
End Sub

Private Sub Combo7_dropdown()
    Static count As Integer
    Me.unit.Clear

    Me.unit.AddItem "baby ear", 0
    Me.unit.AddItem "1/2 cup", 1
    Me.unit.AddItem "cup", 2
    Me.unit.AddItem "large", 3
    Me.unit.AddItem "patoto", 4
    Me.unit.AddItem "pepper", 5
    Me.unit.AddItem "small spear", 6

    If flag(ve) = False Then
        count = 0
        flag(ve) = True
    End If

    count = count + 1
    If count > 5 Then
        re = MsgBox("You can not select more than 5 items.", vbOKOnly)
        Exit Sub
    End If
    i = ve
    j = count

```

```

    Text1.Text = ""
End Sub

Private Sub Command1_Click()

Dim count As Integer, item As Integer

List1.AddItem "You have eaten the following items daily:"

For count = 1 To 6
    For item = 1 To 5
        If qtarray(count, item) <> 0 Then
            List1.AddItem foodarray(count, item) & " " &
qtarray(count, item)
        End If
    Next item
Next count

End Sub

Private Sub Command2_Click()

Load Form1
Form1.Show
Form1.Command2.Enabled = True
Form1.Command1.Enabled = False
Unload Form3

End Sub

Private Sub Command3_Click()
    Dim i As Integer
    Dim l As Integer
    Dim cal1 As Integer
    Dim cal2 As Integer
    Dim cal3 As Integer
    Dim cal4 As Integer
    Dim cal5 As Integer
    Dim cal6 As Integer

    Dim sub1cal As Integer
    Dim sub2cal As Integer
    Dim sub3cal As Integer
    Dim sub4cal As Integer
    Dim sub5cal As Integer
    Dim sub6cal As Integer

    Dim sv1 As Integer
    Dim sv2 As Integer
    Dim sv3 As Integer
    Dim sv4 As Integer
    Dim sv5 As Integer
    Dim sv6 As Integer

    Dim ttcal As Integer, name As String, path As String

```



```

path = Dir1.path
name = path & "\info.txt"

Open name For Output As #1
List1.Clear

sub1cal = 0

For i = 1 To item1.Recordset.RecordCount
  For l = 1 To 5
    If foodarray(gr, l) <> "" Then
      If item1.Recordset.Fields(0).Value & " / " &
        item1.Recordset.Fields(1).Value = foodarray(gr, l) Then
        call = item1.Recordset.Fields(3).Value
        sub1cal = sub1cal + call * qtarray(gr, l)
      End If
    End If
  Next l
  item1.Recordset.MoveNext
Next i

sv1 = sub1cal / 80
List1.AddItem "Your intake for calories and servings are calculated
  according to your input:"
List1.AddItem "grain calories:" & sub1cal
List1.AddItem "grain serving:" & sv1

sub2cal = 0
For i = 1 To item2.Recordset.RecordCount
  For l = 1 To 5
    If foodarray(ve, l) <> "" Then
      If item2.Recordset.Fields(0).Value & " / " &
        item2.Recordset.Fields(1).Value = foodarray(ve, l) Then
        cal2 = item2.Recordset.Fields(3).Value
        sub2cal = sub2cal + cal2 * qtarray(ve, l)
      End If
    End If
  Next l
  item2.Recordset.MoveNext
Next i

sv2 = sub2cal / 25
List1.AddItem "vegetable calories:" & sub2cal
List1.AddItem "vegetable serving:" & sv2

sub3cal = 0
For i = 1 To item3.Recordset.RecordCount
  For l = 1 To 5
    If foodarray(fr, l) <> "" Then
      If item3.Recordset.Fields(0).Value & " / " &
        item3.Recordset.Fields(1).Value = foodarray(fr, l) Then
        cal3 = item3.Recordset.Fields(3).Value
        sub3cal = sub3cal + cal3 * qtarray(fr, l)
      End If
    End If
  Next l
  item3.Recordset.MoveNext

```

```

Next i

sv3 = sub3cal / 60
List1.AddItem "fruit calories:" & sub3cal
List1.AddItem "fruit serving:" & sv3

sub4cal = 0
For i = 1 To item4.Recordset.RecordCount
    For l = 1 To 5
        If foodarray(mea, l) <> "" Then
            If item4.Recordset.Fields(0).Value & " / " &
                item4.Recordset.Fields(1).Value = foodarray(mea, l) Then
                cal4 = item4.Recordset.Fields(3).Value
                sub4cal = sub4cal + cal4 * qtarray(mea, l)
            End If
        End If
    Next l
    item4.Recordset.MoveNext
Next i

sv4 = sub4cal / 55
List1.AddItem "meat calories:" & sub4cal
List1.AddItem "meat serving:" & sv4

sub5cal = 0
For i = 1 To item5.Recordset.RecordCount
    For l = 1 To 5
        If foodarray(mi, l) <> "" Then
            If item5.Recordset.Fields(0).Value & " / " &
                item5.Recordset.Fields(1).Value = foodarray(mi, l) Then
                cal5 = item5.Recordset.Fields(3).Value
                sub5cal = sub5cal + cal5 * qtarray(mi, l)
            End If
        End If
    Next l
    item5.Recordset.MoveNext
Next i

sv5 = sub5cal / 90
List1.AddItem "milk calories:" & sub5cal
List1.AddItem "milk serving:" & sv5

sub6cal = 0
For i = 1 To item6.Recordset.RecordCount
    For l = 1 To 5
        If foodarray(tip, l) <> "" Then
            If item6.Recordset.Fields(0).Value & " / " &
                item6.Recordset.Fields(1).Value = foodarray(tip, l) Then
                cal6 = item6.Recordset.Fields(3).Value
                sub6cal = sub6cal + cal6 * qtarray(tip, l)
            End If
        End If
    Next l
    item6.Recordset.MoveNext
Next i

sv6 = sub6cal / 45

```

```

List1.AddItem "fats & sweets calories:" & sub6cal
List1.AddItem "fats & sweets serving:" & sv6

ttcal = sub1cal + sub2cal + sub3cal + sub4cal + sub5cal + sub6cal
List1.AddItem "Your total intake for calories:" & ttcal & " kcal"

If fg = 1 Then
    Write #1, "grain", 11, sv1
    Write #1, "veg", 5, sv2
    Write #1, "fruit", 4, sv3
    Write #1, "milk", 3, sv5
    Write #1, "meat", 7, sv4
    Write #1, "sugar", 18, sv6
ElseIf fg = 2 Then
    Write #1, "grain", 9, sv1
    Write #1, "veg", 4, sv2
    Write #1, "fruit", 3, sv3
    Write #1, "milk", 3, sv5
    Write #1, "meat", 6, sv4
    Write #1, "sugar", 12, sv6
ElseIf fg = 3 Then
    Write #1, "grain", 6, sv1
    Write #1, "veg", 3, sv2
    Write #1, "fruit", 2, sv3
    Write #1, "milk", 2, sv5
    Write #1, "meat", 5, sv4
    Write #1, "sugar", 6, sv6
End If
Close #1

End Sub

Private Sub Command4_Click()

For i = 1 To 6
    flag(i) = False
    For j = 1 To 5
        foodarray(i, j) = ""
        qtarray(i, j) = 0
    Next j
Next i
List1.Clear
Combo1.Text = "none"
Combo3.Text = "none"
Combo4.Text = "none"
Combo5.Text = "none"
Combo6.Text = "none"
Combo7.Text = "none"
combrnum.Text = ""

item1.Recordset.MoveFirst
item2.Recordset.MoveFirst
item3.Recordset.MoveFirst
item4.Recordset.MoveFirst
item5.Recordset.MoveFirst
item6.Recordset.MoveFirst

```

```

End Sub

Private Sub Form_Load()
Let flag(6) = True
Let flag(1) = True
Let flag(2) = True
Let flag(3) = True
Let flag(4) = True
Let flag(5) = True
Dim name As String, path As String
    path = Dir1.path
    name = path & "\Nutritioninfo.mdb"
    item1.DatabaseName = name
    item2.DatabaseName = name
    item3.DatabaseName = name
    item4.DatabaseName = name
    item5.DatabaseName = name
    item6.DatabaseName = name
End Sub

Private Sub Text1_Change()

qtarray(i, j) = Val(Text1.Text)

End Sub

Private Sub unit_click()

unitarray(i) = unit.Text

End Sub

Form4(IntakeAny.frm)

Dim numitems As Integer, maxdata As Single
Dim label(1 To 6) As String
Dim actual(1 To 6) As Single
Dim stasis(1 To 6) As Single
Dim diff(1 To 6) As Single

Private Sub Command1_Click()

Load Form1
Form1.Show
Form1.Command3.Enabled = True
Form1.Command1.Enabled = True
Form1.Command2.Enabled = True
Unload Me

End Sub

Private Sub Command2_Click()

numitems = 6

Call readdata(label(), actual(), stasis(), numitems, maxdata)

```

```

Call drawAxes(numitems, maxdata)
Call drawdata(actual(), stasis(), numitems)
Call showtitle(maxdata)
Call showlabels(label(), numitems, maxdata)
Call showlegend(maxdata)
Image1.Visible = False
Picture1.Visible = True
Text3.Text = "This graph shows the amount of calories you have taken in
a day. Please click the result button again, you will see another
graph that indicates the improvement you need to take for each items."
End Sub

```

```

Private Sub readdata(label() As String, actual() As Single, stasis() As
Single, numitems As Integer, maxdata As Single)

```

```

Dim i As Integer, path As String, name As String
path = Dirl.path
name = path & "\info.txt"

```

```

Open name For Input As #1
Let maxdata = 0
For i = 1 To numitems
    Input #1, label(i), actual(i), stasis(i)
    If actual(i) > maxdata Then
        Let maxdata = actual(i)
    End If
    If stasis(i) > maxdata Then
        Let maxdata = stasis(i)
    End If
Next i
Close #1

```

```

End Sub

```

```

Public Sub drawAxes(numitems As Integer, maxdata As Single)

```

```

    Picture1.Scale (-1, 1.2 * maxdata)-(numitems + 1, -0.2 * maxdata)
    Picture1.Line (-1, 0)-(numitems + 1, 0)
    Picture1.Line (0, -0.1 * maxdata)-(0, 1.2 * maxdata)

```

```

End Sub

```

```

Public Sub drawdata(actual() As Single, stasis() As Single, numitems As
Integer)

```

```

Dim i As Integer
For i = 1 To numitems
    Let Picture1.FillStyle = 1
    Picture1.Line (i - 0.3, actual(i))-(i, 0), QBColor(0), BF
    Let Picture1.FillStyle = 4
    Picture1.Line (i, stasis(i))-(i + 0.3, 0), QBColor(0), B
Next i

```

```

End Sub

```

```

Public Sub locate(X As Single, Y As Single)

```

```

    Let Picture1.CurrentX = X
    Let Picture1.CurrentY = Y

End Sub

Public Sub showtitle(maxdata As Single)

    Call locate(0.4, 1.2 * maxdata)
    Picture1.FontSize = 10
    Picture1.FontBold = True
    Picture1.Print "Comparing your actual intake with recommended intake"

End Sub

Private Sub showlegend(maxdata As Single)

    Picture1.FillStyle = 1
    Picture1.Line (2, 1.05 * maxdata)-(2.9, 0.95 * maxdata), QBColor(0),
BF
    Call locate(3, 1.05 * maxdata)
    Picture1.Print "Standard"
    Picture1.FillStyle = 4
    Picture1.Line (2, 0.9 * maxdata)-(2.9, 0.8 * maxdata), QBColor(0), B
    Call locate(3, 0.9 * maxdata)
    Picture1.Print "Actual"

End Sub

Public Sub showlabels(label() As String, numitems As Integer, maxdata
As Single)
    Dim i As Integer, lbl As String, lblwid As Single
    Dim lblhght As Single, tickfactor As Single

    For i = 1 To numitems
        Let lbl = label(i)
        Let lblwid = Picture1.TextWidth(lbl)
        Let tickfactor = 0.02 * maxdata
        Picture1.Line (i, -tickfactor)-(i, tickfactor)
        Call locate(i - lblwid / 2, -tickfactor)
        Picture1.Print lbl
    Next i
    Let lbl = Str(maxdata)
    Let lblwid = Picture1.TextWidth(lbl)
    Let lblhght = Picture1.TextHeight(lbl)
    Let tickfactor = 0.01 * numitems
    Picture1.Line (-tickfactor, maxdata)-(tickfactor, maxdata)
    Call locate(-tickfactor - lblwid, maxdata - lblhght / 2)
    Picture1.Print lbl

End Sub

Private Sub Command3_Click()

    Picture1.Visible = False
    Picture2.Visible = True

```

```

Call readin(label(), diff())
Call DrawX
Call drawdiff(diff())
Call showtt
Call showlb(label())
Text3.Text = ""
Text3.Text = "This graph shows how you can improve for your daily
intake. The up and down points indicate the range that your daily
intake should be increase or reduce for each item in servings."

End Sub

Private Sub DrawX()

    Picture2.Scale (-0.2, maxdata + 4)-(numitems + 1, -(maxdata + 4) /
2)
    Picture2.Line (-1, 1)-(numitems + 1, 1)
    Picture2.Line (0, -(maxdata + 4))-(0, maxdata + 4)

End Sub

Private Sub readin(label() As String, diff() As Single)

Dim i As Integer, path As String, name As String
path = Dir1.path
name = path & "\info.txt"

Open name For Input As #1
Let maxdata = 0
For i = 1 To numitems
    Input #1, label(i), stasis(i), actual(i)
    diff(i) = stasis(i) - actual(i)
    If stasis(i) > maxdata Then
        Let maxdata = stasis(i)
    End If
Next i
Close #1
End Sub

Private Sub drawdiff(diff() As Single)

Dim i As Integer
For i = 1 To numitems
    If i < numitems Then
        Let Picture2.DrawStyle = 0
        Picture2.DrawWidth = 2
        Picture2.Line (i, diff(i) + 1)-(i + 1, diff(i + 1) + 1), vbRed
    End If
    Picture2.Circle (i, diff(i) + 1), 0.01 * numitems
    Picture2.Print Str(diff(i))
Next i

End Sub

Private Sub showtt()

```

```

Call location(1.5, maxdata + 1)
Picture2.FontSize = 10
Picture2.FontBold = True
Picture2.Print "Improvement trend for your daily intake "
End Sub

Private Sub location(X As Single, Y As Single)
    Let Picture2.CurrentX = X
    Let Picture2.CurrentY = Y
End Sub

Private Sub showlb(lable() As String)
    Dim i As Integer, lbl As String, lblwid As Single
    Dim lblhght As Single, tickfactor As Single

    For i = 1 To numitems
        Let lbl = label(i)
        Let lblwid = Picture2.TextWidth(lbl) + 2
        Let tickfactor = 0.02 * maxdata
        Picture2.Line (i, -tickfactor + 1)-(i, tickfactor + 1)
        Call locate(i - lblwid / 2, -tickfactor)
        Picture2.Print lbl
    Next i

    Let lbl = Str(maxdata - 2)
    Let lblwid = Picture2.TextWidth(lbl)
    Let lblhght = Picture2.TextHeight(lbl)
    Let tickfactor = 0.01 * numitems
    Picture2.Line (-tickfactor, maxdata)-(tickfactor, maxdata)
    Call locate(-tickfactor - lblwid, maxdata - lblhght / 2)
    Picture2.Print lbl
    Let lbl = Str(-(maxdata - 2))
    Picture2.Line (-tickfactor, -maxdata / 2)-(tickfactor, -maxdata / 2)
    Call locate(-tickfactor - lblwid, -maxdata / 2)
    Picture2.Print lbl
End Sub

```



VITA

Yun Hui Lu

Candidate for the Degree of

Master of Science

Thesis: BUILDING AN EXPERT SYSTEM BY INTERGRATING CLIPS WITH  
VISUAL BASIC

Major Field: Computer Science

Biographical:

Personal Data: Born in Shanghai, the People's Republic of China, August 2, 1964  
The daughter of Jian Yan Lu and Yi Qiao Ni.

Education: Received Bachelor of Science degree in Shanghai Post and  
Telecommunication College, Shanghai, China in July 1983. Completed the  
requirements for the Master of Science degree with a major in Computer  
Science at Oklahoma State University in July, 2000.

Professional experience: Software Engineer, Raymond Karsan Associates,  
Nebraska, USA, September 1999 to January 2000; Office Administrator,  
ARCO Chemical China Ltd. Shanghai Representative Office, Shanghai,  
China, November 1991 to April 1997; Sales and Marketing Representative,  
Shanghai Ocean Hotel, Shanghai, China, March 1989 to May 1991;  
Technical Representative, Shanghai Long Distance Telecommunication  
Bureau, Shanghai, China, August 1983 to February 1989.