Chapter 2 Literature Review

2.1 Introduction to Literature Review

A literature review is a piece of discursive prose and it uses as its database reports of primary or original scholarship, and does not report new primary scholarship itself. The primary reports used in the literature may be verbal, but in the vast majority of cases reports are written documents. The types of scholarship may be empirical, theoretical, critical and analytic, or methodological in nature. A literature review seeks to describe, summarize, evaluate, clarify and or integrate the content of primary reports. Literature Review is an essential part of the whole project. In other word we can say that, it is the backbone of the whole project by devising steps and mechanism to approach the review in a systematic and rigorous way. The main purpose of the literature review is to address the scope, objective and to focus on the project's research domain by performing research and survey to determine and discover the best method required to meet the project's objectives and at the same time making the project a success.

This chapter basically addresses all the common questions asked during development of a project such as:

- What are we trying to achieve in this project?
- How are we going to achieve this?
- Why we need to do this way and not the other way around?

By asking the stated questions at the beginning of the project, we are actually striving to achieve the objective and ensuring we are on the right track. The basic key steps involved in undertaking a literature review that should be followed in sequence are:

- Clarification of the purpose of the literature review in the form of a rationale statement
- Planning the review through drawing up a blueprint document
- Conducting a comprehensive literature search, according to the blueprint
- Selection and focused of reviewing of individual items, according to the blueprint, creating a set of individual reviews
- Integrated or 'synthesis' reviewing according to the blueprint, to produce the review document.

Specifically to this project, this chapter allows to gather the essential and imperative information needed for the development of the tutorial system for C in rule based system architecture. The review of the project actually divided into five main parts:

- General overview of the rule based expert system
- Research on C programming topics, which can be applied rule-based system architecture
- Surveys of existing system and comparison with the new system intended to be developed
- Questionnaire design to get feedback from the students on how it need to be designed
- Overview of the technologies available to ease the development of the system

Source of research varies such as books, questionnaire analysis, existing knowledge based systems, tutorial systems and online articles. Later sections are undoubtedly the outputs of the five key steps mentioned above.

2.2 Introduction to Artificial Intelligence

Artificial Intelligence is a branch of science which deals with helping machines to find solutions to complex problem in a more human-like fashion. There are five branches of AI such as robotics, vision, natural language understanding, sound recognition and knowledge system [2].

Robotics is the study of machines to perform many human-like tasks such as mechanical manipulation and how to make them function with some "intelligence" and autonomy. Robotics is proving to be very valuable to the manufacturing industry. There are robotic automobile painters and assembly line workers.

Vision systems are those that successfully interpret two- and three-dimensional pictures from a two-dimensional image obtained through manmade sensors. This involves processing the image, classifying it and then interpreting the scene.

Natural language is human language. Natural language understanding is the ability to communicate with a computer by conventional language text, such as English and Russian instead of a highly structured language such as standard query language (SQL). Sound recognition is the field that the goal is to enable machines to listen and understand their auditory environment in which processing and reasoning about acoustic sensors such as alarms, spoken words, or automobile engines takes place. Sound recognition systems take an audible sound and make it readable.

Knowledge systems or expert system are software systems that have structured knowledge about a field of expertise. They are able to solve some problems within their

domain by using knowledge derived from experts in the field. This approach emphasizes data interpretation.

2.2.1 Expert System or Knowledge System

The area of knowledge systems has blossomed over the past decade from merely an academic interest into a useful technology. Expert system is one of the most popular and feasible facets of Artificial Intelligence. The most fitting definition of expert system itself is *'a computer program that represents and reasons with knowledge of some specialists subject with a view to solve problems or to give advice'* [6].One of the ways to categorize the application of expert system is a problem-solving paradigm. Experts perform a generic set of tasks when solving certain types of problem such as diagnosis or planning. Regardless of the application area, given the type of problem, the expert collects and reasons with information in similar ways. Expert systems likewise are designed to accomplish generic tasks on the basis of problem type.

It's a software system that has structured knowledge about a field of expertise. It's able to solve some problems within their domain by using knowledge derived from experts in the field. Development of the methods for knowledge representation followed the knowledge acquisition phase. With a suitable amount of knowledge gathered, the structure and representation method for the knowledge system can be described.

2.2.2 The structure of an expert system

The Table 2.1 describes the main structure of an expert system.

Table 2.1: The main structure of expert system

Structure	Description
Knowledge Base	Maintain the expert domain knowledge in a module. Knowledge obtained from the expert is coded here using one of the several knowledge representations.
Working memory	Contain the facts about a problem that are discovered during consultation or inferred by the system.
Inference Engine	Processor to compare and match the facts contained in the working memory with the domain knowledge contained in the knowledge base to conclude the problem.
Explanation Facility	Provides an explanation to the user about why it is asking a question and how it reached some conclusion.

The Figure 2.1 depicts human being's problem solving method. Based on the domain knowledge stored in the long term memory and inferred facts obtained from the short term memory, human could reason a problem and make a conclusion.

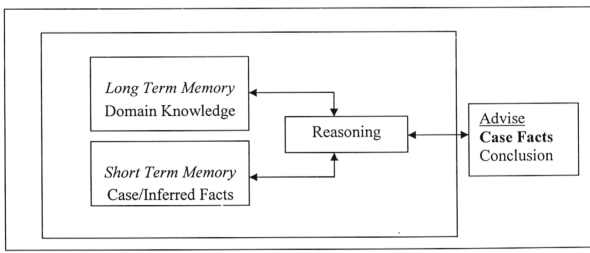

Figure 2.1: Human Expert Problem Solving

Expert system's goal is to enable many people to benefit the knowledge of one person which is the expert. There are two traits of an expert that are attempted to model an expert system which are the expert's knowledge and reasoning. The system must have two principles modules such as a knowledge base and an inference engine to accomplish this. The knowledge base contains highly specialized knowledge on the problem area as provided by the expert. It includes problem facts, rules, concepts and relationship. The inference engine is the knowledge processor, which works as the

reasoning in the human expert problem solving. The knowledge gained through consultation about a problem is stored in the working memory.

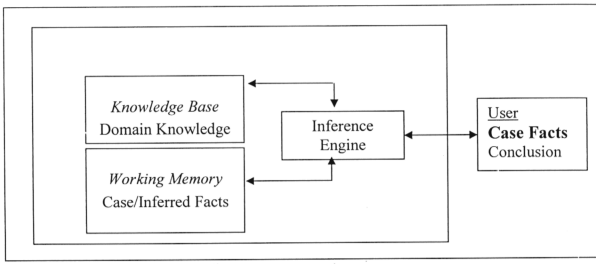

Figure 2.2: Expert System Problem Solving

2.2.3 What is Rule-Based System?

Rule-based system represents problem-solving knowledge as IF_THEN rules. It is one of the oldest techniques for representing domain knowledge in an expert system. It is also one of the most natural and widely used in practical and experimental expert system. Other than the four basic components of an expert system discussed earlier, a rule-based expert system also consists of the following modules:

- **User Interface:** It is the vehicle through which a user views and interacts with the system.
- **External Program:** External program are programs such as spreadsheets and algorithms that work in support for the system.

2.2.3.1 Rule-Based System Architecture

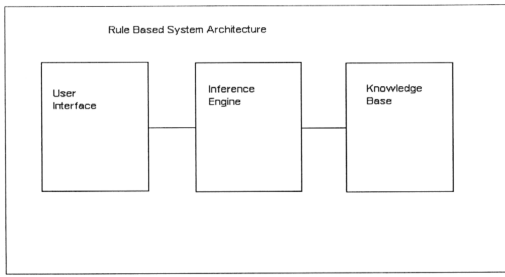

Figure 2.3: Rule Based System Architecture

The above figure shows three major elements of the rule based system architecture, which are the user interface, inference engine and knowledge base. This is the runtime architecture. The user starts the system and interacts with it via the user interface. The engine is the part of the program that actually does stuff. It says, run the system, that is, look at working memory, see what rules fire, and apply them. Actually, same inference engine applies for each rule base. The knowledge base is the rules and the working memory. The rules will remain the same for different runs. Working memory (WM) changes for each run and during the run. Basically working memory is the database, the medium to store the information derived from the system when or during the system run.

The knowledge base consists of rules and working memory (WM). Rules are if and then statements. On the **If** side there will be conditional expressions as (X is green). At the same time, we can have variables in here, in this case X is a variable and on the

Then side, usually have assignments that are set or modify working memory items. Below are some rules:

- if (X is green) and (X is a fruit) then (X is a Watermelon)
- if (X is red) and (X is a fruit) then (X is an Apple)

A real time example:

- **if** (Arithmetic Type EQ **Multiplication)** and (Variable Type EQ **Integer)** and (Variable A Value EQ **A)** and (Variable B Value EQ **B) Then** (int a, b, ans; a=A, b=B, ans =A*B; printf ("%d\n",ans) and Result = A*B;)

2.2.3.2 Inference Mechanisms or Techniques

Expert systems model the reasoning process of humans using technique called inference, which derives new information from known information. In rule-based systems, knowledge is represented as facts about the world and rules to manipulate the facts. At any one time more than one rule may be applied to solve a problem and when each rule is applied other rules may applicable to those rules. Therefore, a rule-based system needs a control structure to decide which rule should be applied first or next and which rules are put together. The two basic inference techniques used in an expert system are forward and backward-chaining. Both techniques are compared to determine the best inference strategy to be adopted into the CTutorial4u project.

Forward-chaining is an inference strategy where conclusions are drawn by first looking at the facts or data on the problem. This style of reasoning is also known as data-driven search. In a rule-based system, forward chaining begins by asserting certain

facts, seeing what rules can fire based on these assertions, picking a rule to fire, then cycles and checks the rules again looking for new matches. This process is continued until a goal is reached or no additional rules can fire.

Backward-chaining is an inference strategy that attempts to prove a hypothesis by gathering supporting information. In a rule-based system, backward chaining begins with a goal and tries to prove it to be true by proving the premises of a rule that contains the goal as its conclusion. The premises of this rule are considered 'sub goals', which the system tries to prove, is true by pursuing other rules that contain the sub goals as conclusions. Eventually, this backward chaining sequence reaches premises that are not supported by other rules and the user is then asked to verify the truth of the premise statement. This type of inference strategy is also called goal-driven search.

In forward chaining, the inferences would be made in the order indicated by the numbers on left flow. Forward chaining uses the depth of the tree structure of condition-action relation. If F was true and a solution before it is inferred that E is true, it may not need to infer that E is true. This forward chaining is used to infer new facts from existing facts. It is also possible to use the same set of rules in reverse to determine what needs to be true for a premise to be true. This method is called backward chaining. Backward chaining is commonly used in rule-based expert systems to enable a hypothesis to be tested and this problem-solving strategy is often referred to as generate and test.

Given an assertion, it is expected that both the assertion and the conclusion are valid throughout the session.

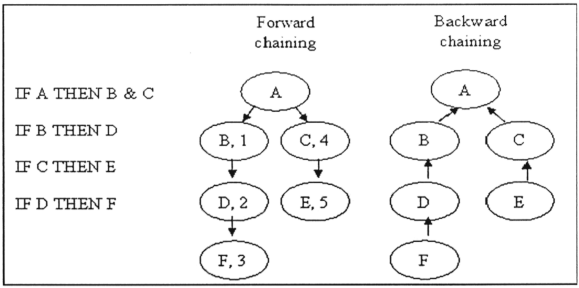

Figure 2.4: Forward and backward chaining

Monotonic reasoning is when a system retains its facts as unchanged assertions. It is more appropriate to be used for a diagnostics and prescription tasks. Non-monotonic reasoning is a method of reasoning that allows for changes in a given fact. Non-monotonic reasoning is an important feature of expert system applied to planning or design tasks.

2.3 Introduction to C Programming

The C programming was developed by Dennis Ritchie at Bell Laboratories in the early 1970s as a system implementation language. From then till now it has evolved into a general-purpose language that combines the convenience of high level languages with the power of assembly language. Currently, standard C compilers are available for many microcomputers, mini computer systems and mainframes. C is becoming increasingly more popular in a variety of computer applications. The language has many powerful features and it is possible to develop portable programs in it and at the same time it is to master and when coupled with good program design techniques it can

be used to generate programs that are well-structured, easy to read and easy to maintain. The C language facilitates a structured and disciplined approach to computer program design.

2.3.1 Basic Structure and Data Types

In general, a C program consists of the following components:

- *Function-main()*
- *Program comments*
- *Preprocessor directives*
- *Data Type declarations*
- *Variables*

Function –main()

Each C program must have one main function. The function main in a program marks the entry point of a program. The opening and closing parentheses () indicate that the identifier is a function. The opening and closing braces {} define the body of the function. The format is

```
main (){
  statements
}
```

Program Comments

Comments are text statements that document and describe the program. Comments are optional, non-executable statements that are placed in the program to explain what the

program does and how the code works. (The computer does not process non-executable statements.) A comment begins with /* and ends with */. The example is as below:

/* A comment may be coded like this */

Preprocessor Directives

A preprocessor directive, also called a compiler directive is an instruction to the preprocessor. The #include preprocessing directive tells the preprocessor to replace the directive with a copy of the file specified by the filename argument within angel brackets <>. The preprocessing directive instructs the preprocessor to modify the source code program. The example is:

include <stdio.h>

According to the example above a copy of the standard input/output header file replaces the directive in the source code. The stdio.h header file enables the program to perform basic input and output operations.

Data Types

Data types stipulates (in what format) the data is stored in memory. In C language, built in data types are classified as fundamental data types and derived data types. Fundamental data types correspond to the most common, fundamental units of a computer and the most common, fundamental ways of using such data. Although C allows other types, these are the only ones commonly being used.

- **int** –to declare numeric program variables of integer types and restricted to whole number such as 5, 16 and 78. Integer variables hold data in the range – 32768 to 32767. If it is beyond or less than the above value then the number is declared using long data type.

- **float**- to declare real or floating point numbers have decimal points such as 2.6571, 74.9 and 567.89.
- **char**- to declare character variables such as single letter, numeric digit, punctuation mark or special symbol. If more than one character, then we define that as string.

Variables

A variable is a data item that may assume different values. Example of how to declare variable is shown below:

int grade;

int final;

The variable grade and final are declared as integers (int). A global variable is declared outside of main and is available to the whole program. A local variable on the other hand is declared inside a specific program function and is not available to any other function.

2.3.2 Topics in C Programming

C programming covers a variety of topics such as arithmetic, control structures, counters, functions, arrays, pointers and file processing. After a thorough analysis of all the topics in C, the above mentioned topics are the topics that will be discussed in detailed in this chapter.

2.3.2.1 Arithmetic

Arithmetic in C is regarding numeric data types and arithmetic calculations. Arithmetic expressions are performed using arithmetic operators. They are (+) for addition, (-) for subtraction, (*) for multiplication, (/) for division and (%) for remainder or modulus.

2.3.2.1.1 Application of rules to teach arithmetic

Based on the literature review, topic arithmetic can be easily taught to the student in by applying rule based system architecture. Based on the question and answer from the system the solution or code for the arithmetic selected is shown or given for the user's view. Figure 2.5 shows the arithmetic for multiplication type which selects the proper code based on the algorithm in the next page. Table 2.2 defines the rules for all the arithmetic types and the algorithm is the same for all arithmetic type.

Q: Problem	**Algorithm**
A: Arithmetic	If Problem is Arithmetic
Q: Types of Arithmetic	Then many types of Arithmetic
A: Multiplication	If type is Multiplication
Q: Variable type?	Then many types of variable type
A: Integer	If variable type is integer
Q: Values of variable A and B	Then value of the variables
A: A is A and B is B	If Value A is A and B is B
	Then choose Code for
	Arithmetic to multiple integer A
	& integer B

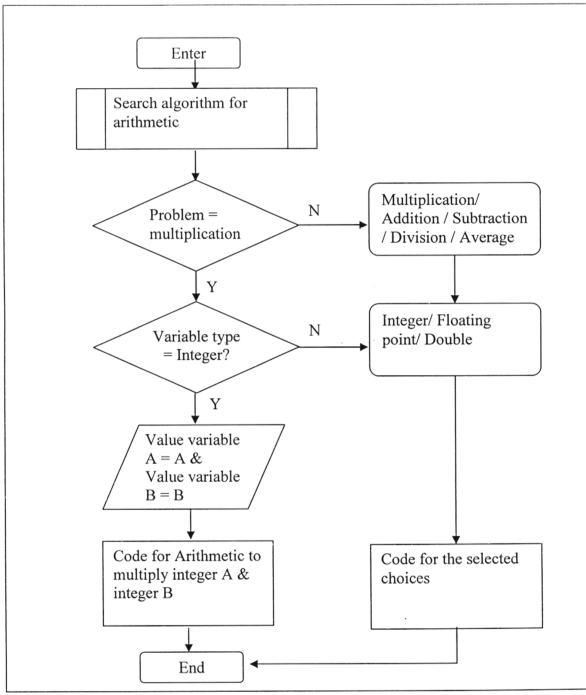

Figure 2.5: Flowchart to teach Arithmetic (multiplication)

Table 2.2: IF-THEN rules for Arithmetic

If (Question & Answer)	Then (Code)
Arithmetic Type EQ **Multiplication** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**int** a, b, ans; **a= A, a=B, ans = A*B;** Printf ("%**d**\n",ans);
Arithmetic Type EQ **Division** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**int** a, b, ans; **a= A, b=B, ans = A/B;** Printf ("%**d**\n",ans);
Arithmetic Type EQ **Addition** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**int** a, b, ans; **a= A, b=B, ans = A+B;** Printf ("%**d**\n",ans);
Arithmetic Type EQ **Subtraction**, AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**int** a, b, ans; **a= A, b=B, ans = A-B;** Printf ("%**d**\n",ans);
Arithmetic Type EQ **Average** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**int** a, b, ans; **a= A, b=B, ans = ((A+B)/2);** Printf ("%**d**\n",ans);
Arithmetic Type EQ **Multiplication** AND Variable Type EQ **Floating point** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**float** a, b, ans; **a= A, b=B, ans = A*B;** Printf ("%**f**\n",ans);
Arithmetic Type EQ **Division** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**float** a, b, ans; **a= A, b=B, ans = A/B;** Printf ("%**f**\n",ans);
Arithmetic Type EQ **Addition** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**float** a, b, ans; **a= A, a=B, ans = A+B;** Printf ("%**f**\n",ans);
Arithmetic Type EQ **Subtraction AND** Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**float** a, b, ans; **a= A, a=B, ans = A-B;** Printf ("%**f**\n",ans);
Arithmetic Type EQ **Average** AND Variable Type EQ **Integer** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**float** a, b, ans; **a= A, b=B, ans = ((A+B)/2);** Printf ("%**f**\n",ans);
Arithmetic Type EQ **Multiplication** AND Variable Type EQ **Double** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**double** a, b, ans; **a= A, b=B, ans = A*B;** Printf ("%**1f**\n",ans);
Arithmetic Type EQ **Division** AND Variable Type EQ **Double** AND Variable A Value EQ A AND Variable B Value EQ **B**	**double** a, b, ans; **a= A, b=B, ans = A/B;** Printf ("%**1f**\n",ans);
Arithmetic Type EQ **Addition** AND Variable Type EQ **Double** AND Variable A Value EQ A AND Variable B Value EQ **B**	**double** a, b, ans; **a= A, b=B, ans = A+B;** Printf ("%**1f**\n",ans);
Arithmetic Type EQ **Subtraction** AND Variable Type EQ **Double** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**double** a, b, ans; **a= A, b=B, ans = A-B;** Printf ("%**1f**\n",ans);
Arithmetic Type EQ **Average** AND Variable Type EQ **Double** AND Variable A Value EQ **A** AND Variable B Value EQ **B**	**double** a, b, ans; **a= A, b=B, ans = ((A+B)/2);** Printf ("%**1f**\n",ans);

2.3.2.2 Control Structures

C has seven control structures all together, namely sequence, three types of selection structure and three types of repetition. The three types of selection structures are **if**, **if/else** and **switch** structure. The if structure is called a single-selection structure because it selects or ignores a single action where it either performs or selects an action if a condition is true or skips the action if the condition is false. On contrary, the if/else selection structure performs an action if a condition is true and performs a different action if the condition is false. The switch selection structure performs one of many different actions depending on the value of the expression.

At the same time, C also provides three types of repetition structure such as **while**, **do/while** and **for**. The while repetition structure allow us to specify that an action is to be repeated while some condition remains true whereas the do/while repetition structure tests the loop continuation condition after the loop body is performed therefore the loop body will be executed at least once. The for loop is to set up a counter-controlled loop. The for statement repeats the statement in the loop a given number of times where the statement body executes as long as the condition test is true.

If Structure

Purpose: To set up one-way conditional branch.

```
if (credits<45){
        printf ("Welcome freshman");
        }
```

If/Else Structure

Purpose: To set up a two-way conditional branch

```
if (credits <45) {
        printf ("Welcome freshman");}
else {
  printf ("Welcome upperclassman");
        }
```

Switch Structure

Purpose: To set up a multi-path conditional path. The switch structure allows the program to select one option from a given set of options.

```
int choice;
.....
printf ( " Enter your choice...\n");
printf ("Movie menu: 1-Action, 2-Comedy, 3-Drama \n");
Switch (choice) {
case 1:
        printf ("Action movie fan\n);
        break;
case 2:
        printf ("Comedy movie fan\n");
        break;
default:
        printf ("Invalid choice\n");
        break;}
```

While Structure

Purpose: To print numbers from 1 to 10.

```
main ()
{
        int counter = 1;
        while (counter <= 10) {
   printf ("%d\n", counter):
                ++counter; }
        return 0;
}
```

Do While Structure

Purpose: To print the numbers from 1 to 10.

```
main() {
        int counter = 1;
        do {
                printf ("%d  ", counter);
                }
        while (++counter <= 10);
        return 0;
}
```

For Structure

Purpose: To print the numbers from 1 to 10.

```
main()
{
int counter;
    for (counter = 1; counter <= 10; counter++)
        printf ("%d\n", countr);
    return 0;}
```

2.3.2.2.1 Application of rules to teach control structures

Based on the analysis, control structures can be taught to the student in an easier way if we apply rule based system architecture. Based on the question and answer from the system the control structure can be designed in rule based system architecture. The correct codes are shown to the users based on the algorithm below. Figure 2.6 shows the control structure for only selection structure and Table 2.3 contains all the if-then rules to be used for control structures.

Q: Problems?	**Algorithm**
A: Control Structure	If Problem is Control Structure
Q: Type of Control Structure	Then many types of Control Structure
A: Selection structure	If type is Selection structure
Q: How many action?	Then number of action to be performed
A: 2	If action is 2
	Then take algorithm for if/else

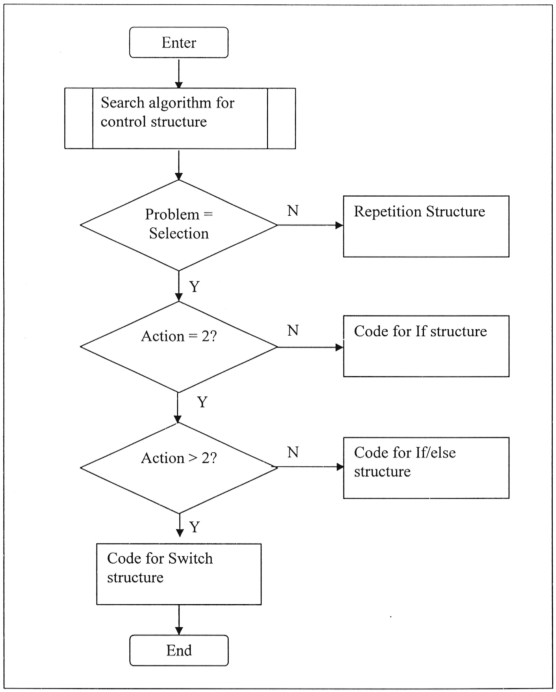

Figure 2.6: Flowchart to teach Control Structure (Selection Structure)

Table 2.3: IF-THEN rules for Control Structure

If (Question & Answer)	Then (Code)
Control Structure EQ **Selection Structure** AND Condition Type EQ **One-Way (If)** AND Grade EQ **A**	**if** grade>= A Then printf("Passed\n", grade);
Control Structure EQ **Selection Structure** AND Condition Type EQ **Two-Way (If/Else)** AND Grade EQ **A**	**if** grade>= A Then printf("Passed\n", grade) **else** printf("Failed"\n", grade);
Control Structure EQ **Selection Structure** AND Condition Type EQ **Multipath (Switch)** AND Grade EQ **A**	**Switch** (Grade/10){ Case 10: Case 9: printf("A:Excellent\n"); Case 8: Case 7: Case 6: printf("B:Credit\n"); Case 5: printf("C:Pass\n"); Case 4: printf("D:OK\n"); Case 3: Case 2: Case 1: Case 0: printf("E:Failed\n"); }
Control Structure EQ **Repetition** Structure AND Condition Type EQ **While** AND Test1 Grade EQ **A** AND Test2 Grade EQ **B** AND Test3 Grade EQ **C** AND Test4 Grade EQ **D** AND Test5 Grade EQ **E**	int counter, grade, total, average; total = 0, counter = 1; **while** (counter<=5){ printf("Enter your grade") scanf("%d", &grade) total = total + grade; counter = counter +1;} average = total /5;
Control Structure EQ **Repetition Structure** AND Condition Type EQ **Do/While** AND Test1 Grade EQ **A** AND Test2 Grade EQ **B** AND Test3 Grade EQ **C** AND Test4 Grade EQ **D** AND Test5 Grade EQ **E**	int counter =1, marks, total=0, average; **do**{ printf("Enter marks") scanf("%d", &marks) total = total + marks; counter = counter +1;} average = total /5; }**while** (++counter<=5)
Control Structure EQ **Repetition Structure** AND Condition Type EQ **While** AND Test1 Grade EQ **A** AND Test2 Grade EQ **B** AND Test3 Grade EQ **C** AND Test4 Grade EQ **D** AND Test5 Grade EQ **E**	int marks, test, average, total=0; **for**(test=1,test<=5,test++) printf("Enter marks") scanf("%d", &marks) total = total + marks; average = total /5;}

2.3.2.3 Functions

Modules in C are called functions [3]. Functions allow programmer to modularize a program. All variables declared in functions definitions are local variables. Most functions have a list of parameters. The parameters provide the means for communicating information between functions. A function's parameters are also local variables. The table below shows the predefined functions in math library [3].

Table 2.4: Examples of math library functions

Function	Description	Example
sqrt(x)	Square root of x	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
exp(x)	Exponential function e^x	exp(1.0) is 2.718282 exp(2.0) is 7.389056
Log(x)	Natural logarithm of x (base e)	Log(2.718282) is 1.0 Log(7.389056) is 2.0

The format of a function definition is:

return-value-type function name (parameter list)

```
{
   declarations
   statements
}
```

When user defines a function, it is called programmer-define function such as function maximum to determine and return the largest of three integers. The example of such function is as below:

```
int maximum (int x, int y, int z)
{
int max = x;
```

```
if (y > max)
    max = y;
if (z > max)
    max = z;
return max;
}
```

2.3.2.3.1 Application of rules to teach function

Although there are many functions in C programming but only function maximum and minimum will provided in this system. This system is only to assist the users to know how the function works and useful to get the answer using numeric calculations.

Q: Problems?	**Algorithm**
A: Function	If Problem is Function
Q: Type of Function	Then many types of Function
A: Maximum	If type is Maximum
Q: Value variable A, B & C?	Then value for variable A and B
A: A= A, B= B & C= C	If B is B and value variable C is C
	Then solution is code for maximum to choose the maximum value between A, B and C.

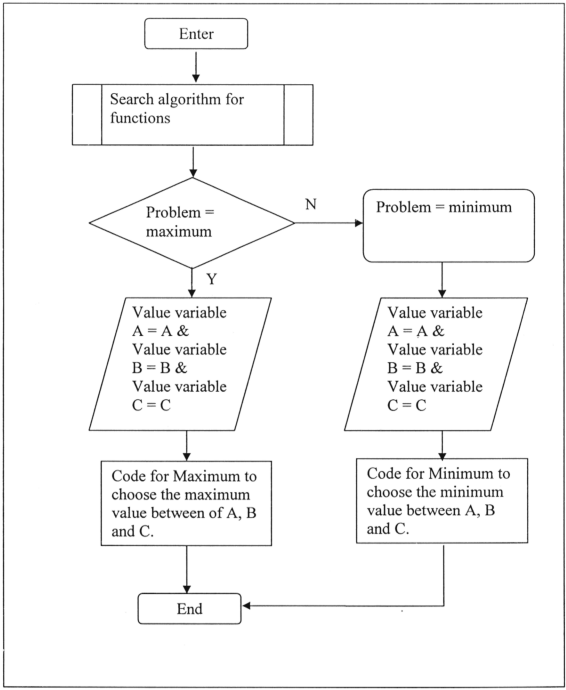

Figure 2.7: Flowchart to teach Functions

Table 2.5: IF-THEN rules for Function

If (Question & Answer)	Then (Code)

Function Type EQ **Maximum** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C**	{int a, b, c;
	printf("Enter integers")
	scanf("%d%d%d",&a,
	&b,&c);
	printf("**Maximum** is: %d,
	Maximum (a,b,c));
	int (int x, int y, int z){
	int ans =w;
	if(x>ans)
	ans=x;
	if(y>ans)
	ans=y;
	if(z>ans)
	ans=z;
	return 0;
	}
	return ans;
	}

Function Type EQ **Minimum** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C**	{int a, b, c;
	printf("Enter integers")
	scanf("%d%d%d",&a,
	&b,&c);
	printf("**Minimum** is: %d,
	Minimum (a,b,c));
	int (int x, int y, int z){
	int ans =w;
	if(x<ans)
	ans=x;
	if(y<ans)
	ans=y;
	if(z<ans)
	ans=z;
	return 0;
	}
	return ans;
	}

2.3.2.4 Arrays and Sorts

Array is a group of memory locations related by the fact that they all have the same name and the same type. The name of the array and the position of the particular element in the array is specified for easy reference. Arrays occupy space in memory. The programmer needs to specify the type of each element and the number of elements required by each array so that the computer may reserve the appropriate amount of memory.

Example:

```
int c [10];     /* to reserve 1o elements for integer array c*/
```

Purpose: To initialize 10 element integer array n to zeros, and prints the array in tabular format.

```
main()
{
        int n[10], I;
        for (i = 0; i<= 9; i++)
                n[i] = 0;
        printf ("%s%13s\n", "Element", "Value");
        for (i=0; i <= 9; i++)
                printf ("%7d%13d\n", I, n[i]);
        return 0;
}
```

Sortings

Sorting is the process of arranging data in a given order. The content of an array may be used to arrange the elements in ascending or descending order. There are three types of sortings namely bubble sort, selection sort and recursive sort.

Example

Purpose: To sort an array of five elements using bubble sort. A list of five elements is arranged in ascending order.

Before sort: 90, 20, 80, 60, 10

After sort: 10, 20, 60, 80, 90

```
for (i = 0; i< 5; i++) {
        if (num[i] > num[i+1]) {
   temp = num[i];
                num[i] = num [i+1];
                num[i +1] = temp;
        }
}
```

2.3.2.4.1 Application of rules to teach arrays

Array can be easily taught to students using rule based system architecture based on the algorithm below.

Q: Problem	**Algorithm**
A: Arrays	If Problem is Arrays
Q: Size of the array	Then there 1 to 10 sizes available to choose
A: 2	If size is 2
Q: Value of integer A and B?	Then values for integer A and B
A: A is A and B is B	If A is A and B is B
Q: Histogram?	Then to display histogram is yes or no
A: Yes	If yes
	Then solution is code for 2 arrays with value A is A, value B is B and show histogram.

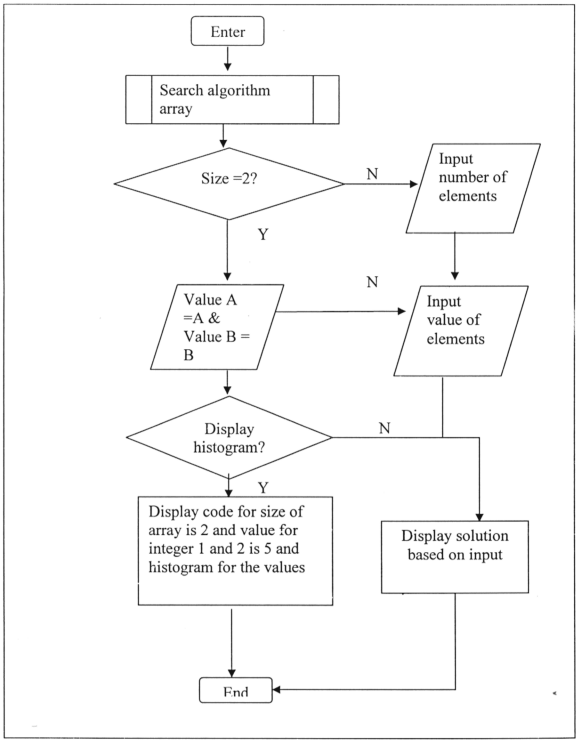

Figure 2.8: Flowchart to teach Arrays

Table 2.6: IF-THEN rules for Array

If (Question & Answer)	Then (Code)

Size of Array EQ **1** AND Value Integer 1 EQ **A** AND Histogram EQ **N**	#define **SIZE 1**
	int main (){
	int n [SIZE] = {**A**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **2** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Histogram EQ **N**	#define **SIZE 2**
	int main (){
	int n [SIZE] = {**A, B**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **3** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Histogram EQ **N**	#define **SIZE 3**
	int main (){
	int n [SIZE] = {**A,B,C**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **4** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Histogram EQ **N**	#define **SIZE 4**
	int main (){
	int n [SIZE] = {**A,B,C,D**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **5** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Histogram EQ **N**	#define **SIZE 5**
	int main (){
	int n [SIZE] = {**A,B,C,D,E**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Table 2.6, continued

If (Question & Answer)	Then (Code)

Size of Array EQ **6** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ E AND Value Integer 6 EQ **F** AND Histogram EQ **N**	#define **SIZE 6**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **7** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Histogram EQ **N**	#define **SIZE 7**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **8** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Histogram EQ **N**	#define **SIZE 8**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G,H**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **9** AND Value Integer 1 EQ AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Value Integer 9 EQ **I** AND Histogram EQ **N**	#define **SIZE 9**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G,H,I**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Size of Array EQ **10** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Value Integer 9 EQ **I** AND Value Integer 10 EQ **J** AND Histogram EQ **N**	#define **SIZE 10**
	int main (){
	int n [SIZE] =
	{**A,B,C,D,E,F,G,H,I,J**};
	int i;
	printf ("%s%13s\n", " Element", "Value");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	return 0;}

Table 2.6, continued

If (Question & Answer)	Then (Code)

Size of Array EQ 1 AND Value Integer 1 EQ **A** AND Histogram EQ **Y**	#define **SIZE 1**
	int main (){
	int n [SIZE] = {**A**};
	int i;
	printf (“%s%13s\n”, “ Element”, “Value”, “**Histogram**”);
	for (i=0, i<=SIZE -1; i++) {
	printf(“%7d%13d “,i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf(“%c”\n, ‘*’)
	return 0;}

Size of Array EQ **2** AND Value Integer 1 EQ **A** ANDValue Integer 2 EQ **B** AND Histogram EQ **Y**	#define **SIZE 2**
	int main (){
	int n [SIZE] = {**A,B**};
	int i;
	printf (“%s%13s\n”, “ Element”, “Value”, “Histogram”);
	for (i=0, i<=SIZE -1; i++) {
	printf(“%7d%13d “,i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf(“%c”\n, ‘*’)
	return 0;}

Size of Array EQ **3** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Histogram EQ **Y**	#define **SIZE 3**
	int main (){
	int n [SIZE] = {**A,B,C**};
	int i;
	printf (“%s%13s\n”, “ Element”, “Value”, “Histogram”);
	for (i=0, i<=SIZE -1; i++) {
	printf(“%7d%13d “,i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf(“%c”\n, ‘*’)
	return 0;}

Size of Array EQ **4** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Histogram EQ Y	#define **SIZE 4**
	int main (){
	int n [SIZE] = {**A,B,C,D**};
	int i;
	printf (“%s%13s\n”, “ Element”, “Value”, “Histogram”);
	for (i=0, i<=SIZE -1; i++) {
	printf(“%7d%13d “,i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf(“%c”\n, ‘*’)
	return 0;}

Table 2.6, continued

If (Question & Answer)	Then (Code)

Size of Array EQ **5** AND Value Integer 1 EQ **A** Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Histogram EQ **Y**	#define **SIZE 5**
	int main (){
	int n [SIZE] = {**A,B,C,D,E**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

Size of Array EQ **6** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Histogram EQ **Y**	#define **SIZE 6**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

Size of Array EQ **7** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Histogram EQ **Y**	#define **SIZE 7**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

Size of Array EQ **8** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Histogram EQ **Y**	#define **SIZE 8**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G,H**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

Table 2.6, continued

If (Question & Answer)	Then (Code)

Size of Array EQ **9** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Value Integer 9 EQ **I** AND Histogram EQ **Y**	#define **SIZE 9**
	int main (){
	int n [SIZE] = {**A,B,C,D,E,F,G,H,I**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

Size of Array EQ **10** AND Value Integer 1 EQ **A** AND Value Integer 2 EQ **B** AND Value Integer 3 EQ **C** AND Value Integer 4 EQ **D** AND Value Integer 5 EQ **E** AND Value Integer 6 EQ **F** AND Value Integer 7 EQ **G** AND Value Integer 8 EQ **H** AND Value Integer 9 EQ **I** AND Value Integer 10 EQ **J** AND Histogram EQ **Y**	#define **SIZE 10**
	int main (){
	int n [SIZE] =
	{**A,B,C,D,E,F,G,H,I,J**};
	int i;
	printf ("%s%13s\n", " Element", "Value", "**Histogram**");
	for (i=0, i<=SIZE -1; i++) {
	printf("%7d%13d ",i, n[i]);}
	for (j=1, j<=n[i], j++)
	printf("%c"\n, '*')
	return 0;}

2.3.2.5 Pointers

Pointers are variables that contain memory addresses as their values. Normally a variable directly contains a specific value whereas a pointer contains an address of a variable that contains a specific value. Pointers must be declared before they can be used such as in the example below:

Example:

int *countPtr, count;

The pointer operators are & and *. The & or address operator is a unary operator that returns the address of its operand. The example of algorithm is as given below:

int y = 5;

int *yPtr;

```
yPtr = &y;          /* assigns the address of the variable y to pointer variable yPtr */
```

The * operator or commonly referred as the indirection operator or dereferencing operator, returns the value of the object to which its operand points.

Example:

```
printf ("%d', *yPtr);     /* points the value of variable y */


main ()
{
        int a;
        int *aPtr;
        a=7;
        aPtr = &a;
        printf ("The address of a is %p\n"
    "The value of aPtr is %p\n\n", &a, aPtr);
        printf ("The value of a is %d\n"
    "The value of *aPtr is %d\n\n", a, *aPtr);
        printf("Proving that * and & are complements of"
    "each other.\n&*aPtr = %p\n*&aPtr = %p\n",
    &*aPtr, *&aPtr);
        return 0;
}
```

2.3.2.5.1 Application of rules to teach pointers

Q: Problem	**Algorithm**
A: Pointers	If Problem is Pointers
Q: Types of pointer solutions	Then many types of pointer solutions
A: Call by Reference	If solution type is call by reference
Q: Variable value?	Then value of variable
A: A	If variable value is 5
	Then take algorithm for Pointer and call by reference for value variable is 5

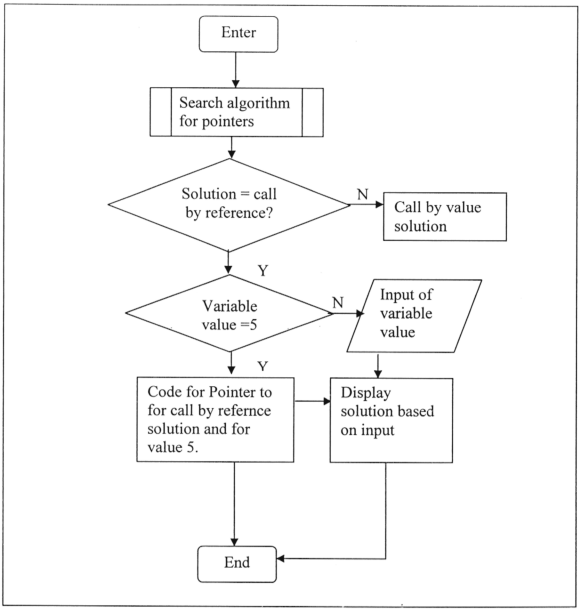

Figure 2.9: Flowchart to teach Pointers

Table 2.7: IF-THEN rules for Pointer

If (Question & Answer)	Then (Code)

Ways to pass argument EQ **Call by Value** AND Function Type EQ **Cube the Value** AND Number of Variable EQ **One** AND Value Integer A EQ **A**	int **cubebyvalue(int);**
	main(){
	int number = **A**;
	printf("Original value %d\n",number);
	number =cubebyvalue(number);
	printf("New value of the number %d\n", number);
	return 0;}
	int **cubebyvalue(int n)** {
	return n * n* n;
	}

Ways to pass argument EQ **Call by Value** AND Function Type EQ **Square the Value** AND Number of Variable EQ **One** AND Value Integer A EQ **A**	int **squarebyvalue(int);**
	main(){
	int number = **A**;
	printf("Original value %d\n",number);
	number = squarebyvalue(number);
	printf("New value of the number %d\n", number);
	return 0;}
	int squarebyvalue(int n) {
	return n * n;
	}

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value** AND Number of Variable EQ **One** AND Value Integer A EQ **A**	int **cubebyvalue(int*)**;
	main(){
	int number = **A**;
	printf("Original value %d\n",number);
	number = cubebyvalue(number);
	printf("New value of the number %d\n", number);
	return 0;}
	int **cubebyvalue(int *nPtr)** {
	return *nPtr * *nPtr * *nPtr);
	}

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Square the Value** AND Number of Variable EQ **One** AND Value Integer A EQ **A**	int **squarebyvalue(int*);**
	main(){
	int number = A;
	printf("Original value %d\n",number);
	number = squarebyvalue(number);
	printf("New value of the number %d\n", number);
	return 0;}
	int squarebyvalue(int *nPtr) {
	return *nPtr * *nPtr;
	}

Table 2.7, continued

If (Question & Answer)	Then (Code)
Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **2** AND Value Integer A EQ **A**, Value Integer B EQ **B** AND Sort EQ **N**	#define **SIZE 2** {int a [SIZE] = {**A,B**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Square the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **3** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Sort EQ **N**	#define **SIZE 3** {int a [SIZE] = {**A,B,C**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference**, Function Type EQ **Square the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **4**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Sort EQ **N**	#define **SIZE 4** {int a [SIZE] = {**A,B,C,D**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference**, Function Type EQ **Square the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **5**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Sort EQ **N**	#define **SIZE 5** {int a [SIZE] = {**A,B,C,D,E**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference**, Function Type EQ **Square the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **6**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Value Integer F EQ **F**, Sort EQ **N**	#define **SIZE 6** {int a [SIZE] = {**A,B,C,D,E,F**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Square the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **7** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Value Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Value Integer G EQ **G** AND Sort EQ **N**	#define **SIZE 7** {int a [SIZE] = {**A,B,C,D,E,F,G**}; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}

Table 2.7, continued

If (Question & Answer)	Then (Code)
Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Square the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **8**, Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Value Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Value Integer G EQ **G** AND Value Integer H EQ **H** AND Sort EQ **N**	#define **SIZE 8** {int a [SIZE] = **{A,B,C,D,E,F,G,H}**; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference**, Function Type EQ **Square the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **9**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Value Integer F EQ **F**, Value Integer G EQ **G**, Value Integer H EQ **H**, Value Integer I EQ **I**, Sort EQ **N**	#define **SIZE 9** {int a [SIZE] = **{A,B,C,D,E,F,G,H,I}**; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}
Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Square the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **10** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Value Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Value Integer G EQ **G** AND Value Integer H EQ **H** AND Value Integer I EQ **I** AND Value Integer J EQ **J** AND Sort EQ **N**	#define **SIZE 10** {int a [SIZE] = **{A,B,C,D,E,F,G,H,I,J}**; int i; printf("Data item in original order\n"); **for(i=0, i<SIZE, i++)** **printf("%4d\n", a[i]);** return 0;}

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **2** AND Value Integer A EQ **A AND** Value Integer B EQ **B** AND Sort EQ **Y**	#define **SIZE 2**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = **{A,B}**;
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr;
	*element1Ptr = hold;}

Table 2.7, continued

If (Question & Answer)	Then (Code)

Ways to pass argument EQ **Call by Reference** AND Function Type **EQ Square the Value** AND Number of Variable EQ **More Than One** AND Number of Variable EQ **3** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Sort EQ **Y**	#define **SIZE 3**
	void bubbleSort(int*, const int)
	{int a [SIZE] = {**A,B,C**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr; *element1Ptr = hold;}

Ways to pass argument EQ **Call by Reference**, Function Type EQ **Cube the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **4**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Sort EQ **Y**	#define **SIZE 4**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = {**A,B,C, D**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr; *element1Ptr = hold;}

Ways to pass argument EQ **Call by Reference**, Function Type EQ **Square the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **5**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Sort EQ **Y**	#define **SIZE 5**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = {**A,B,C,D,E**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr; *element1Ptr = hold;}

Table 2.7, continued

If (Question & Answer)	Then (Code)

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value**, AND Number of Variable EQ **More Than One** AND Number of Variable EQ **6** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** ANDValue Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Sort EQ **Y**	#define **SIZE 6**
	void bubbleSort(int*, const int)
	{int a [SIZE] = {**A,B,C, D,E,F**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void swap(int*, int*)
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr;
	*element1Ptr = hold;}

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value**, AND Number of Variable EQ **More Than One** AND Number of Variable EQ **7** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C** AND Value Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Value Integer G EQ **G** AND Sort EQ **Y**	#define **SIZE 7**
	void bubbleSort(int*, const int)
	{int a [SIZE] = {**A,B,C,D,E,F,G**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr;
	*element1Ptr = hold;}

Table 2.7, continued

If (Question & Answer)	Then (Code)

Ways to pass argument EQ **Call by Reference**, Function Type EQ **Cube the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **8**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Value Integer F EQ **F**, Value Integer G EQ **G**, Value Integer H EQ **H**, Sort EQ **Y**	#define **SIZE 8**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = {**A,B,C, D,E,F,G, H**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = ***element1Ptr**;
	*element1Ptr= ***element2Ptr**;
	*element1Ptr = hold;}

Ways to pass argument EQ **Call by Reference**, Function Type EQ **Cube the Value**, Number of Variable EQ **More Than One**, Number of Variable EQ **9**, Value Integer A EQ **A**, Value Integer B EQ **B**, Value Integer C EQ **C**, Value Integer D EQ **D**, Value Integer E EQ **E**, Value Integer F EQ **F**, Value Integer G EQ **G**, Value Integer H EQ **H**, Value Integer I EQ **I**, Sort EQ **Y**	#define **SIZE 9**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = {**A,B,C,D,E,F,G,H,I**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void **swap(int*, int*)**
	int pass,j;
	for **(pass =0; pass < size-1; pass++)**
	if (array[j]> array[j+1])
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = ***element1Ptr**;
	*element1Ptr= ***element2Ptr**;
	*element1Ptr = hold;}

Table 2.7, continued

If (Question & Answer)	Then (Code)

Ways to pass argument EQ **Call by Reference** AND Function Type EQ **Cube the Value** AND Number of Variable EQ **More Than One**, AND Number of Variable EQ **10** AND Value Integer A EQ **A** AND Value Integer B EQ **B** AND Value Integer C EQ **C**, Value Integer D EQ **D** AND Value Integer E EQ **E** AND Value Integer F EQ **F** AND Value Integer G EQ **G** AND Value Integer H EQ **H**, AND Value Integer I EQ **I** AND Value Integer J EQ **J** AND Sort EQ **Y**	#define **SIZE 10**
	void **bubbleSort(int*, const int)**
	{int a [SIZE] = {**A,B,C,D,E,F,G,H,I,J**};
	int i;
	printf("Data item in original order\n");
	for(i=0, i<SIZE, i++)
	printf("%4d\n", a[i]);
	return 0;}
	void **bubbleSort (int *array, const int size)**
	void swap(int*, int*)
	int pass,j;
	for (pass =0; pass < size-1; pass++)
	if **(array[j]> array[j+1])**
	swap (&array[j], &array[j+1]);}
	void swap (int *element1Ptr, int *element2Ptr)
	int hold = *element1Ptr;
	*element1Ptr= *element2Ptr;
	*element1Ptr = hold;}

2.3.2.6 File Processing

Storage of data in variables and arrays is temporary; all such data is lost when a program terminates. Files are used for programmer's retention of large amounts of data. Computers store files on secondary storage devices, especially disk storage devices. Files can be created, updated and processed. This part will explain further on how to create a sequential file.

2.3.2.6.1 Application of rules to teach file processing

File processing can be taught easily using rule-based system architecture based on the algorithm below. Figure 2.10 explains how the system give the output of code based on the user selection and Table 2.8 depicts the if-then rules for file processing.

Q: Problem

A: File Processing

Q: Types of processes ?

A: Create A Sequential File

Q: Variable type to be written?

A: Integer

Q: Variable name?

A: A

Q: File name?

A: test

Algorithm

If Problem is File Processing

Then many types of processes

If process is create a sequential file

There are many variable types

If variable type is integer

Then variable name

If variable name is A

Then file name

If file name is test

Then code for create sequential file and integer A and file name is test.

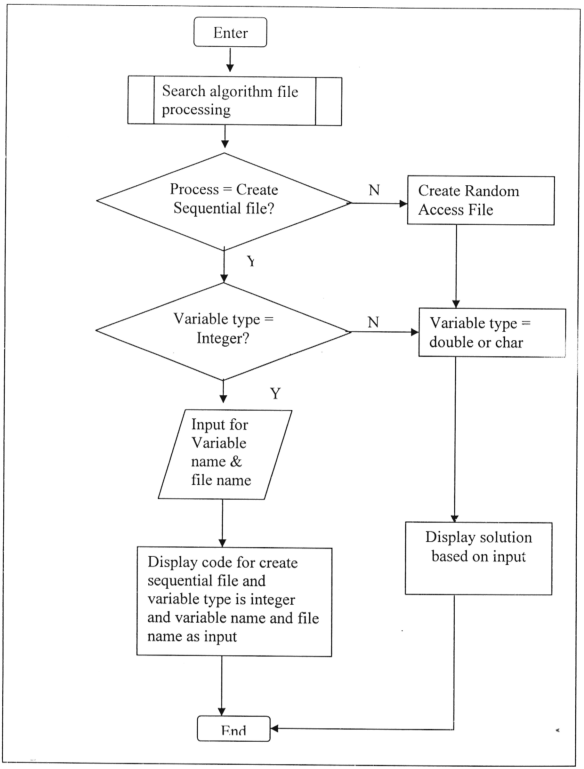

Figure 2.10: Flowchart to teach File processing (Create Sequential File)

Table 2.8: IF-THEN rules for File Processing

If (Question & Answer)	Then (Code)

Process EQ **Create Sequential File** AND
Variable Type EQ **Integer** AND
Variable A Value EQ **A**
	{**int A;**
	FILE *cfPtr;
	if((cfPtr = fopen ("A.text", "w"))==NULL)
	printf("File could not be opened\n")
	else
	printf("Enter the A\n")
	printf("Enter EOF to end input\n")
	printf("Enter your data here\n")
	scanf("%d, &A)
	while (!feof(stdin)){
	fprintf(cfPtr, "%d\n,"A)
	scanf("%d, &A)}
	fclose (cfPtr);}
	return 0;}

Process EQ **Create Sequential File** AND
Variable Type EQ **Double** AND
Variable A Value EQ **A**
	{**double A;**
	FILE *cfPtr;
	if((cfPtr = fopen ("A.text", "w"))==NULL)
	printf("File could not be opened\n")
	else
	printf("Enter the A\n")
	printf("Enter EOF to end input\n")
	printf("Enter your data here\n")
	scanf("%1f, &A)
	while (!feof(stdin)){
	fprintf(cfPtr, "%d\n,"A)
	scanf("%1f, &A)}
	fclose (cfPtr);}
	return 0;}

Process EQ **Create Sequential File** AND
Variable Type EQ Char AND
Variable A Value EQ **A**
	{char A;
	FILE *cfPtr;
	if((cfPtr = fopen ("A.text", "w"))==NULL)
	printf("File could not be opened\n")
	else
	printf("Enter the A\n")
	printf("Enter EOF to end input\n")
	printf("Enter your data here\n")
	scanf("%c, &A)
	while (!feof(stdin)){
	fprintf(cfPtr, "%d\n,"5)
	scanf("%c, &A)}
	fclose (cfPtr);}
	return 0;}

2.3.3 Benefits of Designing in Rule-Based System Architecture

Ctutorial4u is designed in rule-based system architecture in order to develop the best tutoring system where there will be input from user to retrieve solution from the system. It is a two-way learning process. The system is designed with respect to user interaction where the system acts as a guide for the students to excel in C programming for novice users.

2.3.4 Analysis and Synthesis

Based on the expert system explanation and its components, rule based system architecture is chosen to develop Ctutorial4u as it is closely resembles the way human experts solve problem.

Forward chaining

Forward chaining can provide a considerable amount of information from only a small amount of data. It works well when a problem naturally begins by gathering information and seeing what can be inferred from it. This is what needed in Ctutorial4u where it is more feasible to begin a session with data given by the user to derive conclusion rather than beginning a session by trying to prove a present conclusion is valid.

Monotonic Reasoning

Monotonic reasoning is the best reasoning technique for Ctutorial4u since the facts remain unchanged in the database for the problems. The rules are set in the inference engine to for look appropriate solution.

2.4 Surveys on existing system

Thousands of expert systems have been constructed in the last few years. Expert systems have been applied in many areas, such as business, chemistry, education, finance, law, mathematics, medicine, mining and space technology. They are used for control, design, diagnosis, prediction, planning, simulation etc. Thousands of systems have been developed and are in use throughout the world. Expert systems have moved from the research labs to the general market place and industry. This has resulted from the better understanding of the technology and the production of tools for building such systems. The Table 2.9 shows some of the early expert system constructed.

Table 2.9: Example of expert systems

Expert system	Description
DENDRAL	A system developed at Stanford to interpret mass spectrograms.
Drilling Advisor	A system developed by Elf to help determine why a drill sticks.
MYCIN	A medical diagnosis system.
XCON	A computer configuration system developed by DEC for VAX computers.
LENDING ADVISOR	Used for evaluating the risks on possible loans.
PROUST	For finding semantic bugs in novice Pascal programmers' code.

We need to analyze some similar system to get some ideas, knowledge and guidance. So, there are few system being analyzed and compared to produce a system which meets the requirements of users.

2.4.1 MYCIN

Mycin was an expert system developed at Stanford in the 1970s using Lisp. It is a program for advising physicians on treating bacterial infections of the blood and meningitis. MYCIN conducts a question and answer dialog. After asking basic facts about the patient such as name, sex and age, MYCIN asks about suspected bacterial organisms, suspected sites of infection, the presence of specific symptoms such as fever or headache which is relevant to diagnosis. It then recommends a certain course of antibiotics. MYCIN's dialogs are in English so it avoids having to understand freely written English by controlling the dialog. It outputs sentences, but the user types only single words or standard phrases. Its major innovations over many previous expert systems were that it uses measures of uncertainty (not probabilities) for its diagnoses and the fact that it is prepared to explain its reasoning to the physician, so he can decide whether to accept it.

MYCIN extended the notion that the knowledge base should be separate from the inference engine, and its rule-based inference engine was built on a backward-chaining or goal-directed control strategy. Since it was designed as a consultant for physicians, MYCIN was given the ability to explain both its line of reasoning and its knowledge. Mycin represented its knowledge as a set of IF-THEN rules with certainty factors. The following is an English version of one of Mycin's rules:

IF the infection is pimary-bacteremia

AND the site of the culture is one of the sterile sites

AND the suspected portal of entry is the gastrointestinal tract

THEN there is suggestive evidence (0.7) that infection is bacteroid.

The 0.7 is roughly the certainty that the conclusion will be true given the evidence. If the evidence is uncertain, the bits of certainties of evidence will be combined with the certainty of the rule to give derive a conclusion. The action part of the rule could just be a conclusion about the problem being solved, or it could be an arbitrary lisp expression. This allowed great flexibility.

Mycin use the basic backward chaining reasoning strategy that we described above. However, Mycin used various heuristics to control the search for a solution or proof of some hypothesis. These helped to make the reasoning efficient and to prevent the user being asked too many unnecessary questions.

Though it's an expert system for medical field and not for education or learning, but MYCIN act as a premier or role-model to rely on for all future expert systems development. The rules IF- THEN of MYCIN undoubtedly are the guidance for CTutorial4u.

2.4.2 The Rule-Based Expert System Using an Interactive QA Sequence

This rule-based system consists largely of a main window system (knowledge acquisition module, rule-based inference engine, and user interface for input/output), a simple GIS mapping system, and a database [2]. The knowledge acquisition module, inference engine, and user interface were built using Visual Basic 6.0. The GIS mapping system was also built using Visual Basic 6.0 based on MapObjects Version 2. MapObjects has many GIS facilities and can be extended and integrated easily with other systems using a conventional language such as Visual C++ and Visual Basic. The database for the knowledge base (prologue, questions and answers, and rules) is designed in Microsoft Access 2002.

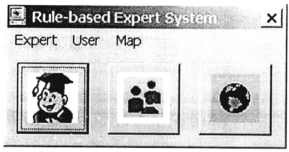

Figure 2.11: Main menu of rule-based expert system

Expert main menu: Consists of three sub menus: new problem, open problem. The expert part is the knowledge acquisition module. The expert part can be used to construct new knowledge (prologue, question, answer, and rule) or to update the existing knowledge by experts.

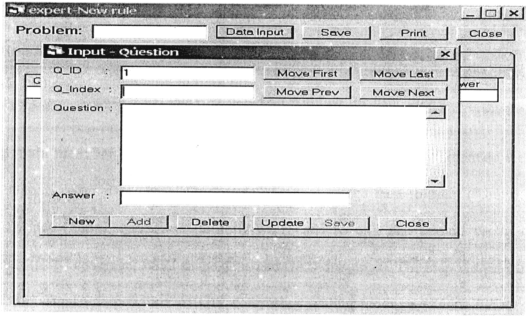

Figure 2.12: Expert menu's input window

User main menu: The rule-based inference engine was implemented in the user part because the inference rule and the search strategy are needed in the user part in order to solve a selected problem. This inference engine is built based on deductive reasoning using forward chaining. The inference engine will generate questions automatically when the user selects a problem. The solution will also be generated from the previous questions and answers using the existing rules by the rule-based inference engine.

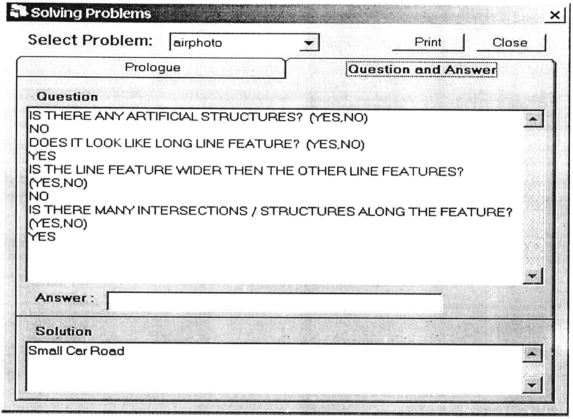

Figure 2.13: Solving problem window for user menu

Map main men: Map main menu has sub menu called map viewer. This map viewer can support raster data (ESRI Grid data), standard image data (bitmap (*.bmp), gif (*.gif), jpeg (*.jpg), window metafile (*.wmf), and so on), and several vector data formats (ESRI coverage, ESRI shape data, and CAD drawings). The map viewer can be used for visual interpretation of GIS data using scaling modules (Zoom In/Out and Pan). The map viewer also can be used for the thematic mapping. The thematic mapping process has four steps. First, the user selects features on the existing image or map. Second, the user draws polygons for the boundary of the selected features. Third, the user interprets the selected features using the user part in the expert system. Last, the user labels the polygon for selected features using labeling module when the expert system generates the solution.

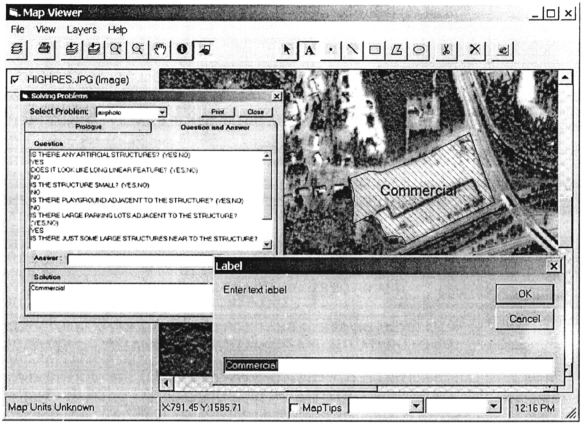

Figure 2.14: Map Viewer of Rule-Based Expert System

Table 2.10: Rules for Rule-Based Expert System

If	Then
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ NO AND SITUATION2 EQ NO	Residential
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ NO AND SITUATION2 EQ YES	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ YES	Industrial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ YES AND SITUATION2 EQ NO	School
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ YES AND SITUATION2 EQ YES	Service (park)
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ NO	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ NO AND PATTERN EQ NO	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ NO AND PATTERN EQ YES	Golf Course
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ YES AND SITUATION4 EQ NO	Residential (Apartment)
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ YES AND SITUATION4 EQ YES	Industrial
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ YES	Highway
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ NO AND SITUATION5 EQ NO	Railroad
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ NO AND SITUATION5 EQ YES	Small Car Road
STRUCTURE EQ NO AND PATTERN EQ YES	Golf Course
STRUCTURE EQ NO AND PATTERN EQ NO AND SHAPE4 EQ YES	Lake
STRUCTURE EQ NO AND PATTERN EQ NO AND SHAPE4 EQ NO AND TONE EQ WHOLEY	Evergreen

The Rule-Based Expert System discussed earlier actually is quite similar to CTutorial4u. The If-Then rules in Table 2.10 actually are guideline to develop the rules for CTutorial4u. This system uses interactive question and answer session of YES and NO answers to derive a solution. There are so many ways to derive solution.

CTutorial4u can't use question and answer with YES and NO answer but it is designed to use question and answer with selection criteria. The answers are set in the rules so the users can only select the answers specified.

2.4.3 INTELLITUTOR II

An Intelligent programming environment for learning programming is an interactive application of knowledge-based Artificial Intelligence [11]. INTELLITUTOR II consists of GUIDE, ALPUS II and a C environment as an integrated environment, and is implemented as a server-client system on the Internet. The server system consists of a Web server, an ALPUS II server and a C server [10]. The service includes reading C documents, editing C programs with guide and help functions, detecting logical bugs by means of a knowledge-based program understander ALPUS II, compiling the program and executing it with the C environment. ALPUS is a knowledge-based program understander by means of four kinds of programming knowledge on program semantics, which are knowledge on algorithms, knowledge on programming techniques, knowledge on variables and knowledge on bugs. The knowledge on a programming language is used as well as a base level understanding.

Knowledge Modeling in ALPUS II

In ALPUS II the algorithm-oriented programming knowledge plays a key role in understanding a buggy program. This knowledge is represented in a hierarchical data structure called HPG.

Program Understanding in ALPUS II

Program understanding in APLUS II is done by four major steps such as [9]:

Step 1: Generalization of program statements is done first, mainly by introducing the language independent representation of program statements in AL (abstract language). AL was designed to represent source statements in both Pascal and C, so that knowledge-based program comprehension can be done by means of the common knowledge base in which every peace of the knowledge is represented in the AL formalism. Student's program is translated to AL statements first in this step. For example, an assignment statement is represented as "(<- Left-variable Expression)", and conditional loops are represented as "(?Loop Expression Statement)" for a so-called pre-check loop for a While-Do statement in Pascal and a While statement in.

Step 2: Normalization of program segments is done next by five steps to decrease a variety of program code. At first, a source program in AL is converted to a Lisp-like representation with line number. Next, order of relational operators is normalized so that the evaluation can be easier. Simplification of a program structure is done next. For example, meaningless procedures are detected and inserted into calling program by modification of code and variables. Deletion of redundant type and constant declarations is done next. Deletion of the RECORD-type declarations is done lastly.

Step 3: Identification of key variables and their roles is done next based on the data obtained by Cognitive experiment as shown in the following procedure. At first rough segmentation of the source program is done by means of HPG-oriented algorithm knowledge. Then, likelihood scores are evaluated by means of the associated attributes attached to each key variable stored within the knowledge base, where the role and locations

within the algorithm are defined. The variable which has been assigned with the highest score is known as the identified variable and its role. The instance frame is generated from the associated class frame for each identified variable. It should be stressed that in this process bugs are not considered. This strategy works very well even if the program includes bugs and incompleteness.

Step 4: Identification of each process in detail is done next as the final step. The logical bugs and associated intentions are identified in this step by pattern matching techniques between the student's program segments and the template knowledge. The pattern matching is done against each node of the HPG graph from the root to the leaves one-by-one as follows:

1) One process is picked up from 5 the HPG nodes.

2) The template is generated from the attached knowledge and associated identified variables.

3) The pattern matching is applied to an associated segment of the target student's program to detect correctness or buggyness. The standard pattern is tried first. If the template matches then the system understands this segment as correct. If failed, then acceptable patterns are tried one by one. If one of the acceptable patterns has matched then the system supposes that although this segment would work well better coding should be used instead. If failed again, then buggy patterns are tried. If one of them matched then the system supposes this as a buggy segment. If failed, then the system supposes this segment as buggy one. However any advice can not be produced since information is not available within the knowledge base.

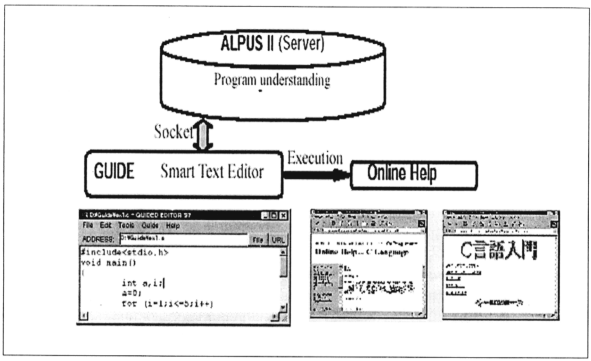

Figure 2.15: Overview of the APLUS II

2.4.4 The ANDES Physics Tutoring System

The Andes project originated with an Office of Naval Research management initiative to forge close relationships between ONR and the Navy's academic institutions. In particular, there was an interest in trying out artificially intelligent tutoring technology, a longstanding research area for ONR, at the Naval Academy. The Andes system is an intelligent tutoring system that has helped hundreds of students to improve their learning for physics in university. It replaces pencil and paper problem solving homework. Students continue to attend the same lectures, labs and recitations. Five years of experimentation at the United States Naval Academy indicates that it significantly improves student learning [7]. This report is a comprehensive description of Andes. It describes Andes' pedagogical principles and features, the system design and implementation, the evaluations of pedagogical effectiveness, and our plans for dissemination.

Students read the problem (top of the upper left window), draw vectors and coordinate axes (bottom of the upper left window), define variables (upper right window) and enter equations (lower right window). These are actions that they do when solving physics problems with pencil and paper. As soon as an action is done, Andes gives immediate feedback unlike Pencil and Paper Homework (PPH). Entries are colored green if they are correct and red if they are incorrect. This is called flag feedback. In Figure 2.16, all the entries are green except for equation 3, which is red. Variables are defined by filling out a dialogue box, such as one shown in Figure 2.17. Vectors and other graphical objects are first drawn by clicking on the tool bar on the left edge of Figure 2.16, then drawing the object using the mouse, then filling out a dialogue box like the one in Figure 2.17. Filling out these dialogue boxes forces students to precisely define the semantics of variables and vectors. PPH does not require this kind of precision, so students often just use variables in equations without defining them. If students include an undefined variable in an Andes equation, the equation turns red and a message box pops up indicating which variable(s) are undefined.

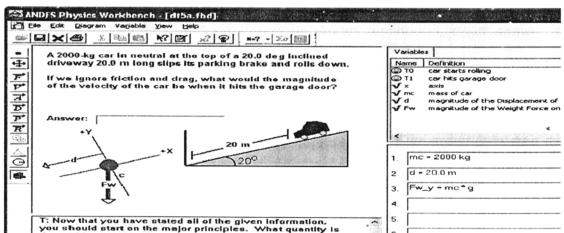

Figure 2.16: Andes screen

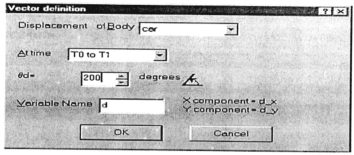

Figure 2.17: A dialogue box for drawing a vector

2.4.5 Softsyst

Softsys provides software services as well as training courses. It is an online system. The C++ Quiz is used heavily from visitors worldwide which consists of short (12 questions) or detailed (55 questions).

The main feature of the system is the C++ quiz is which has two portion short quiz and detailed quiz. Basically here, the idea and design of the quiz is being evaluated to get better idea how the new system can be designed. The quiz is designed as a collection of question from various topics in C++ language and it is not based on topics which it is not a good learning approach provided to naïve users but it is a good practice for those who are in the intermediate stage. The intention of the system is to provide chance to the intermediate user test their knowledge in various topics of C++ to master the language and acts a testing ground of knowledge. The system applies good design principles of human computer interaction. It has a smiley face to indicate the percentage of correct answer out of the total questions and if is beyond 50% then the smiley face will appear with smiling face whereas if the marks are lower than 50% a sad face will appear. These are not only the features and another main feature is the solution for the wrong answers. The system is able to detect the wrong answers and mange to give the correct answers with reasons stated.

The principles of C++ quiz design will help to design the interface for new rule-based system for the Ctutorial4u.

2.4.6 Carnegie Learning Cognitive Tutor Integrated Math I, II, III

Using Cognitive Tutor Integrated Math curricula, students work with multiple representations of a linear function: tables, graphs, algebraic formulas and written text. By translating from one representation to another, students gain a solid understanding of how the representations interconnect [2]. Cognitive Tutor let the students learn in an environment that is similar to working one-on-one with an instructor. Each student benefits from an individualized course of instruction complete with immediate feedback, prescriptive mediation and assessment.

- **Problem Scenario:** The Problem Scenario helps students make connections between real-life problem situations and the mathematics needed to solve them. Embedded questions provide breadth and relevance to the problem-solving process.
- **Worksheet:** As students progress through the curriculum, they learn to generalize specific instances into mathematical formulas. Students complete the Worksheet (which functions like a spreadsheet) by recording answers to questions posed in the Problem Scenario.
- **Solver:** The Solver encourages students to solve equations within the context of the problem. Students learn techniques to solve problems and discover the value of mathematical skills beyond the classroom.
- **Graph:** Students represent mathematical functions graphically, set boundaries and intervals, label axes and plot points and lines.

Just-in-Time Help Messages: When students make errors, they receive immediate feedback. This gives students the opportunity to correct mistakes quickly. Teachers can spend more time with students who need additional help, confident that all students are engaged and on task.

Skills: The Cognitive Tutor dynamically assesses and tracks each student's progress and level of understanding on specific mathematical skills. As the Tutor guides them down an individualized learning path, students can access this information on demand, which encourages them to be accountable for their own learning progress.

Though it is not designed for C programming language but it has some basic good principles of designing a tutorial system for the students such the problem scenario, worksheets, solver, just-in time help messages and skills as mentioned above. The only drawback of the system is it is not designed in a knowledge based system where **if then rule** statement was not being used. The suggestion to this system is to use rule-system architecture to enhance it functionality, modifiability and scalability.

2.5 CTutorial4u Vs Existing Systems

The analysis of other system is very important to develop a new system although the existing system does not perform all the functionalities as the new system but actually the new system will adopt all the good strategies in the similar existing system and try to eliminate all the lacking in terms of design and functionalities.

Table 2.11: Comparison of existing system's and functionalities

System / Functionalities	MYCIN	Rule-Based Expert System	INTELLITUTOR	ANDES	Softsys	Cognitive Tutor
If-Then rules	Y	Y	N	N	N	N
Tutorial	N	N	Y	Y	Y	Y
Quiz	N	N	N	N	Y	Y
Help	Y	Y	Y	Y	Y	Y
Human computer interaction metaphor	Y	Y	Y	Y	Y	Y
Central database	Y	Y	Y	Y	Y	Y
User friendliness	N	Y	Y	Y	Y	Y
Consistency	Y	Y	Y	Y	Y	Y
Security	Y	Y	Y	Y	Y	Y
Immediate feedback	Y	Y	Y	Y	Y	Y
Program compiler/solver	Y	Y	Y	Y	Y	Y

The table above explains that all the systems functionalities were equally contributed to the system design of CTutorial4u. All these systems actually lacking at least one of the functionalities but the Ctutorial4u complying all the above functionalities and attempts to give the best to the users by adopting all good behaviors. The non rule-based expert systems such as Softsys and Cognitive Tutor didn't use If-Then rules such as INTELLITUTOR and ANDES (intelligent systems) whereas Softsys and Cognitive Tutor (normal online systems). Human computer metaphor is one of the interactivity features needed by expert system which complied by Rule-Based Expert System Using Interactive Question and Answer Squence, ANDES, Softsys and Cognitive Tutor such as using the traffic light and smiley concepts in quiz session. All the systems uses central database and all has consistent interface throughout the whole system. These systems are user friendly where users don't find it difficult to use the systems the second time they login in except MYCIN needs the expert in that field who are in medical line to use because mostly uses medical terms. In terms of security, all the

system allows the administrators to amend the database or the design of the system where all are well protected with password.

After analyzing all the above stated system's functionalities, the Ctutorial4u includes just-in time help messages from Cognitive Tutor to indicate the percentage of correct answer out of the total questions and if is beyond 50% then the message excellent will appear with smiley whereas if the marks are lower than 50% then a try again message will appear with sad smiley. Ctutorial4u basically adopts this feature from Cognitive Tutor. Apart from that, CTutorial4u not only act as learning software but it is more like a real tutor where based on the user selection for every chapter, the system somehow solve the user directed problems using rules in the working memory. The memory contains all the possible solution for every topic in C.

On top of that, CTutorial4u also include quiz for every chapter to test the student's knowledge about a particular topic and store the respective student's marks in the database and quiz session applies the human computer metaphor adopted from Cognitive Tutor. All the systems compared above, provide immediate feedback to students and not like normal procedures in classroom or hospital where we got to wait till the lecturer mark our papers and doctors to diagnose the sickness. The only drawback of CTutorial4u is it does provide C compiler within the system itself for the student to test their programming skills or ability to write codes on their own where with it the system actually can correct their errors by debugging their codes.

2.6 Software and Technologies

Visual Basic 6.0 will be used to design the user interfaces and Microsoft Access 2000 to design the database of CTutorial4u.

2.6.1 Visual Basic

Visual Basic 6.0 is one the programming language used to develop most of the standalone software. It is based on graphical and event-driven user interface where an object can be built easily sing interface and the codes are easy to built as well. Actually event processor controls Visual Basic and nothing will happen until the event is being identified. When the event is identified, code relevant with the event procedure will be processed and done.

Advantages of Visual Basic

- Perfect set of objects
- Many icons and reusable components inclusive of graphical interface
- Good design of data structure for mathematic and string and graphic function operation
- Effectively connected to database especially (Open Database Connectivity) – server/client based for Microsoft SQL Server, SyBaseSQL and Oracle and Microsoft Access 2000 as well.
- Supports ActiveX
- Use package and deployment wizard to differentiate all the application easily
- Ability to connect to Internet

Added features in Visual Basic 6.0

- The Scripting Runtime Library
- The FileSystemObject and TextStream classes
- The Dictionary class
- New language features
- Dynamic control creation

- The CallByName feature
- The new Extender Validate event and CausesValidation property for controls
- Arrays can be returned from functions and assigned
- New component creation features
- Use UDTs as parameters or return types of public classes
- Persist class data in ActiveX components
- CreateObject improvement
- A class can act as a DataSource
- A class can act as a DataConsumer
- New HasDC property
- New members of the UserControl class
- Controls can be LightWeight (windowless)
- New FontChanged event of the StdFont class

2.6.2 Microsoft Access

Database Management System (DBMS) actually provide access to the users to reach the data and to change the data to relevant information needed by the users. There are many available DBMS but Microsoft Access 2000 is used to develop Ctutorial4u. Microsoft Access is the best way to connect intranet to retrieve data from database fast and effectively.