Software base for reuse-oriented program development.

Deoraj. Ramjisingh

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/3507

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S’il manque des pages, veuillez communiquer avec l’université qui a conféré le grade.

La qualité d’impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l’aide d’un ruban usé ou si l’université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d’auteur, SRC 1970, c. C-30, et ses amendements subséquents.
Software Base for Reuse-Oriented
Program Development

by
Deoraj Ramjisingh

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the
University of Windsor

Windsor, Ontario, Canada
1994
THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

ISBN 0-612-01479-7
Abstract

Software reusability, the construction of systems by the usage of knowledge or artifacts associated with existing systems, has been of significant importance to both engineers and managers. As the reusable software components are growing, we are faced with the dilemma of storing these components so as to allow easy retrieval for reuse. Many multifaceted approaches to automatic retrieval of reusable software components have been devised and implemented. Unfortunately they all fail to place any emphasis on the structuring of the software repository so as to provide for easy retrieval.

In this thesis we investigate the problem of structuring the large and evolving software component repository to provide an effective way to store and ultimately locate/retrieve reusable software components, as an aid in the first two phases of reuse-oriented program development. The components structuring was done based on the polymorphic and extra-argument relations among components. Added to the retrieval of software components formed of an exact match between query type and library types, these relations can lead to the retrieval of software components whose types are more general, specific or contains more argument than the queried type. A prototype system was designed and developed. Applications of the system was described for functional programming.
To my mother Rabinandanie
my father Jeet Narain
my sister Lindy &
my brothers Rudy, Dale, Denny, Prits, Raj
Acknowledgments

I would like to acknowledge the support and guidance provided by Dr. Y. Park, whose time, dedication and effort has contributed in guiding me through this thesis. Without his patience and active supervision it would have been impossible to complete it within this time frame. I am grateful to Dr. R. Frost for providing valuable suggestions and comments that aided in my progress. I would also like to thank Dr. H. K. Kwan for his willingness to be a part of this work by serving as an external reader.

Special thanks to my parents for always emphasizing the importance of knowledge and for always being there for me. Their constant support and encouragement has contributed significantly towards my successful completion.

Last but not least, I would like to thank all my colleagues in the grad lab for providing a warm and friendly atmosphere throughout my Masters program, and to let them know, they are indeed true friends.
TABLE OF CONTENTS

Abstract ................................................................. iv
Acknowledgments ....................................................... vi
List of Figures .......................................................... x
List of Tables ........................................................... xiii
1 Introduction ............................................................ 1
  1.1 Software Reuse .................................................... 1
  1.2 Functional Programming .......................................... 3
    1.2.1 Polymorphic Functions ....................................... 3
    1.2.2 Higher-Order Functions ..................................... 6
  1.3 Overview of the Thesis ........................................... 7
    1.3.1 Motivation .................................................... 8
    1.3.2 Our Approach: A Structured Repository .................... 9
  1.3.3 Thesis Statements ............................................. 12
  1.3.4 Organization of Thesis ....................................... 13
2 Software Base: A Structured Approach to Repository .......... 14
  2.1 Functional Generalization ...................................... 14
    2.1.1 General Functions .......................................... 14
    2.1.2 Functions with more Arguments ............................ 16
  2.2 Functional Specialization ...................................... 18
  2.3 Structuring of Software Components Repository ............ 19
    2.3.1 Polymorphic Relation ...................................... 19
    2.3.2 Extra-Argument Relation ................................... 21
<table>
<thead>
<tr>
<th>Appendix A</th>
<th>Sample Of Software Library</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1  One-Argument file</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>A.2  Two-Argument file</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>A.3  Three-Argument file</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Program Listing</td>
<td>73</td>
</tr>
<tr>
<td>Appendix C</td>
<td>User Manual</td>
<td>148</td>
</tr>
<tr>
<td>C.1  Main Window</td>
<td></td>
<td>148</td>
</tr>
<tr>
<td>C.2  Retrieval</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>C.2.1 Various Matches</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>C.3  Insertion</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>C.4  Deletion</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>C.5  Browser</td>
<td></td>
<td>160</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
<td>163</td>
</tr>
<tr>
<td>VITA AUCTORIS</td>
<td></td>
<td>165</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1    Software construction ........................................ 8
Figure 2    The use of types as search keys .......................... 11
Figure 3    Use of general function. .................................. 16
Figure 4    Use of freezing argument function. ...................... 17
Figure 5    Sample polymorphic relation between types. ............ 19
Figure 6    A type with many parents and children ................. 20
Figure 7    Sample extra argument relation between types .......... 21
Figure 8    Sample 1 of software repository ........................ 22
Figure 9    Sample 2 of software repository ........................ 23
Figure 10   Reduction of links based on extra-argument relation ... 24
Figure 11   Overview of the WISER .................................. 28
Figure 12   Insertion .................................................. 30
Figure 13   Insertion of root records ................................. 31
Figure 14   Insertion of \([a] \rightarrow b\) into one-argument structure .... 32
Figure 15   Insertion of \(a \rightarrow [b]\) into one-argument structure .... 32
Figure 16   Insertion of \([(a)] \rightarrow b\) into one-argument structure ... 32
Figure 17   Insertion of \([(a)] \rightarrow [b]\) into one-argument structure ... 33
Figure 18   One-argument structure before insertion of \(a \rightarrow a\) .... 34
Figure 19   Incorrect insertion of \(a \rightarrow a\) into one-argument structure . 34
Figure 20   Correct insertion of \(a \rightarrow a\) into one-argument structure . 35
Figure 21   Insertion of type \([a] \rightarrow [a]\) into one-argument structure .... 35
Figure 22   Ripple effect of inserting \((=) : a \rightarrow a \rightarrow \text{bool}\) ........... 38
Figure 23   Ripple effect of inserting mem : \(a \rightarrow [a] \rightarrow \text{bool}\) ......... 38
Figure 24  Ripple effect of inserting member :: (a → b → bool) → a
— [b] → bool ..................................................... 40
Figure 25  Retrieval .................................................. 41
Figure 26  Example of retrieval .................................. 44
Figure 27  Deletion ..................................................... 45
Figure 28  Organization of windows .............................. 46
Figure 29  Main menu window .................................... 47
Figure 30  Retrieval dialog for a function of type [a] → a .... 48
Figure 31  Dialog window for exact match ..................... 49
Figure 32  Dialog window for more specific functions ........ 50
Figure 33  Dialog window for extra-argument functions ...... 50
Figure 34  Dialog window for more general functions ....... 51
Figure 35  Exact match ............................................... 52
Figure 36  General functions ........................................ 52
Figure 37  Extra-argument functions ............................ 53
Figure 38  Insertion dialog for inserting concat :: a → [a] → [a] . 54
Figure 39  Insertion error dialog ................................. 54
Figure 40  Deletion dialog for deleting concat :: a → [a] → [a] . 55
Figure 41  Deletion error dialog1 ................................. 55
Figure 42  Deletion error dialog2 ................................. 55
Figure 43  Browser window ........................................ 56
Figure 44  The prime numbers building block diagram ...... 57
Figure C.1  Main Window ........................................... 149
Figure C.2  Retrieval dialog ....................................... 151
Figure C.3  Retrieval confirmation dialog ..................... 152
| Figure C.4 | No exact match dialog | 152 |
| Figure C.5 | Exact match dialog | 153 |
| Figure C.6 | Specific function dialog | 154 |
| Figure C.7 | General function dialog | 154 |
| Figure C.8 | Extra-argument function dialog | 155 |
| Figure C.9 | Insertion dialog | 156 |
| Figure C.10 | Insertion Conformation dialog | 157 |
| Figure C.11 | Insertion error dialog | 157 |
| Figure C.12 | Deletion dialog | 158 |
| Figure C.13 | Deletion conformation dialog | 159 |
| Figure C.14 | No name error | 160 |
| Figure C.15 | No type match error | 160 |
| Figure C.16 | Number of argument for browser dialog | 160 |
| Figure C.17 | Browser window | 162 |
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Type constructor</td>
<td>4</td>
</tr>
<tr>
<td>Table 2</td>
<td>Monomorphic types</td>
<td>4</td>
</tr>
<tr>
<td>Table 3</td>
<td>Polymorphic types</td>
<td>5</td>
</tr>
<tr>
<td>Table 4</td>
<td>Higher-order Functions</td>
<td>7</td>
</tr>
<tr>
<td>Table 5</td>
<td>The sum function</td>
<td>7</td>
</tr>
<tr>
<td>Table 6</td>
<td>Possible Instances</td>
<td>15</td>
</tr>
<tr>
<td>Table 7</td>
<td>Record structure</td>
<td>24</td>
</tr>
<tr>
<td>Table 8</td>
<td>Algorithm for insertion of all possible types based on argument combinations</td>
<td>37</td>
</tr>
<tr>
<td>Table 9</td>
<td>Algorithm for updating a record extra-argument field when an exact type match is found</td>
<td>39</td>
</tr>
<tr>
<td>Table 10</td>
<td>Exact match retrieval algorithm</td>
<td>42</td>
</tr>
<tr>
<td>Table 11</td>
<td>Magic square with $m = 5$</td>
<td>60</td>
</tr>
<tr>
<td>Table 12</td>
<td>A Program that solves the magic square problem</td>
<td>61</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction

1.1 Software Reuse

Since, from the beginning of stored-program computers in 1949, software construction was obtained by either starting from scratch and writing the code line by line or by reusing existing software components. However, the late 1960s saw the birth of the "software crisis", which is still prevalent today. This can be attributed to the problem of software demand growing exponentially and software developers not being adequately prepared, with the relevant tools to handle such an immense increase.

"Software reuse is the process of using existing software artifacts rather than building them from scratch" [6]. Reuse covers a wide range of products, including documentation, source-code fragments, design structures, analysis, specification and module-level implementation structures, just to name a few.

Reuse-oriented software development is a process that is decomposed into three distinct phases. The first phase deals with the storage of components in the software repository. Developers need to have a complete understanding of each component in the software repository, especially if it has to be modified. This is accomplished by having an abstraction associated with each component, which allows the developers access to important information while hiding unimportant ones. Without this, reusability would be difficult, since the developer has to know what each component does, along with when and how to reuse them. The second phase focuses on the location/retrieval aspect as related to finding components. This involves finding exact or very similar components to the one needed, so that minimum modification can be done if need be. In order to identify

---

1 EDSAC was the first stored-program computer invented and developed at the University of Cambridge in 1949.[16]
such a component, an understanding of all the components in the software repository is required. The final phase deals with the adaptation/integration of the components once they are found.

There exist two main approaches to location/retrieval of suitable reusable software components. The first can be termed syntactic approach which deals with the syntax of the components and is really retrieval by informal specification. It uses keywords to aid in the location/retrieval phase of reuse-oriented program development. However, to use this technique the system users are required to have knowledge of both the structure of the software base and that of the keyword set. So, one can readily see that this approach demands too much from the users. Next there is the semantic approach which deals with the meaning of components. Under this approach we have three techniques for retrieval. The first is retrieval based on formal specification which although ideal for reuse proves to be generally undecidable and hence unimplementable. It is undecidable because it is impossible to determine whether two functions are equivalent, since there are infinite number of properties to test for, leading to the halting problem. However, if the domain was restricted, then it might be possible to decide whether they are equivalent, hence being able to implement. So a restricted form of specification is therefore implemented. The second is retrieval based on Sampling Behavior[8] which deals with retrieval of executable components. Basically the user supplies a set of input data and their resulting output. The technique then determines suitable candidates for reuse based on this sampling data, by executing the components in the library with the input data and comparing the results with the known output results. Finally, we have retrieval based on types. This technique although less accurate than formal specification, gives more useful information than the syntactic approach. The user of such a system would query for reusable components based upon their types as search keys[11, 12, 13].
1.2 Functional Programming

"Programming in a functional language consists of building definitions and using the computer to evaluate expressions"[1]. Programmers in such a language are primarily responsible for the construction of a function that will ultimately solve the problem at hand. This function in turn may be solved by having a number of intermediate or subsidiary functions. Functions in functional languages may be passed as arguments to other functions, returned by functions as results, and be included in lists, tuples, etc.[4]. In all functional languages there exist the concept of polymorphic functions and higher-order functions, both of which greatly enhances the language’s ability to perform the required programming task. Polymorphic functions and higher-order functions can be credited with the ability to perform the same tasks as many different functions. These features will be explained further in subsequent sections.

1.2.1 Polymorphic Functions

Simply put a type can be considered as a simple form of specification. But, for one to understand types more fully consistency in notations must be brought to the foreground. Operators like num (number type), char (character type) and bool (boolean type) are the three well known primitive types that will be considered through out this thesis, along with the type constructor ―→ (function type), ―,” (pair type) and ―‖ (list type). The type constructors are used to form more complex types, as shown in Table 1, by applying them to an appropriate number of type arguments. The variables a, b, c, .... are called type variables, which means that they are unknown types, that are still to be determined. So we can now say a type is either a type variable, a primitive type or a complex type.

Types containing no type variables are called monomorphic types. Thus the primitive types are monomorphic, as well as the types of many primitive functions such as sum,
which finds the sum of a list of numbers and `conj`, which does the boolean conjunction of two boolean values. Table 2 shows these functions along with their associated types.

Types containing one or more type variables are called *polymorphic types*\cite{15, 7, 3}. The word polymorphic is of Greek origin and means to have “many forms”. Therefore functions having polymorphic types can perform the same operation on several different data types.

The functions `fst`, which returns the first component of a tuple and `drop`, which drops the first \( n \) element from a list of elements, are both polymorphic since they operate on data that could have many different types. Table 3 shows these functions along with their associated types.

A type is considered to be shallow if it is of the form \( \forall a \forall b \ldots \forall n. \alpha \), where \( \alpha \) is a polymorphic type containing possible type variables \( a, b, c, \ldots, n \) and no nested quantifiers\cite{10}. Simply speaking, type variables are quantified at the top level. Thus,
Table 3 Polymorphic types

\[ \begin{align*}
\text{fst} & \quad : (a, b) \rightarrow a \\
fst \ (x, y) & = x \\
\text{drop} & \quad : \text{num} \rightarrow [a] \rightarrow [a] \\
drop \ 0 \ xs & = xs \\
drop \ (n+1) \ [] & = [] \\
drop \ (n+1) \ (x:xs) & = \text{drop} \ n \ xs
\end{align*} \]

\[ \forall a \forall b (a \rightarrow b) \text{ and } \forall a ([a] \rightarrow \text{num}) \] can be termed shallow types. Non-shallow types on the other hand do allow nested quantifiers, such as \( (\forall a, a) \rightarrow (\forall b, b) \).

Let's for a brief moment consider types with their associated quantifiers. Thus, if for the type \( \forall a (a \rightarrow a) \) we want to instantiate \( a \) with \( \forall b, b \rightarrow \text{char} \), care must be taken so that we avoid non-shallow types. To do this we will have to settle for less general instances which are shallow, thereby disallowing substitutions which yields \( \forall b, b \rightarrow \text{char} \rightarrow (\forall b, b \rightarrow \text{char}) \), while accepting the appropriate shallow type \( \forall b, b \rightarrow \text{char} \rightarrow (b \rightarrow \text{char}) \). This thesis does not deal with non-shallow types since no inference type checking algorithm is available. Generally, quantifiers will be omitted from types since they are implicit (i.e \( \forall a \forall b (a \rightarrow b) \equiv (a \rightarrow b) \)).

In the epicenter of software reuse, the retrieval of components that does not form exact matches are of great importance. These components are usually immediate or less than immediate instances of the required one, based upon their types. So, it is necessary to determine whether the type of a function \( f :: \alpha \) is an instance of the type of a function \( g :: \beta \), that is, \( \alpha \prec \beta \) (\( \prec \) is used to define one type as an instance of another). Formally, type \( \alpha \) is an instance of type \( \beta \), if \( \alpha \) can be obtained from \( \beta \) by relevant linear consistent substitution of type for previously defined type variables that occurs in the type \( \beta \). As such

1. \( ([\text{num}] \rightarrow (\text{num}, b) \rightarrow \text{num}) \prec ([a] \rightarrow (a, b) \rightarrow \text{num}) \), by the substitution of \text{num} for type variable \( a \),
2. \([num] \rightarrow (num, num) \rightarrow num\) \(\prec \) \([a] \rightarrow (a, b) \rightarrow num\). by the substitution of 

*num* for type variables *a*, *b*, and 

3. \([(t, u, v)] \rightarrow ((t, u, v), b) \rightarrow num\) \(\prec \) \([a] \rightarrow (a, b) \rightarrow num\), by the substitution of 

\((t, u, v)\) for type variable *a*.

The instance relation is a partial order, which means it’s reflexive, transitive and antisymmetric.

The unification of types are also of significant importance because it is necessary to determine whether two types are unifiable. Formally, types \(\alpha\) and \(\beta\) are unifiable, if they have a common instances. For example; if

\[
\begin{align*}
\alpha &:: a \rightarrow a & \text{and} \\
\beta &:: (b, \text{int}) \rightarrow c 
\end{align*}
\]

then, if we substitute \((b, \text{int})\) for *a* and \((b, \text{int})\) for *c* we will get the unified type \((b, \text{int}) \rightarrow (b, \text{int})\), hence making the two types unifiable.

**1.2.2 Higher-Order Functions**

"A function which takes a function as argument, or delivers one as result, is called a higher-order function"[1]. In functional programming, higher-order functions are considered an important asset because if applied well they can simulate (mimic) a number of other functions, thereby reducing the size of programs.

*map* and *foldr1* are examples of two very important higher-order functions (see Table 4). They can be used in many different ways to aid in the construction of other functions. Suppose we are interested in constructing a function that finds the sum of all the numbers in a list of numbers. Then by using the higher-order function *foldr1* we can indeed construct such a function. Table 5 shows how this is done. A closer examination
Table 4 Higher-order Functions

map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = f x : map f xs

foldr1 :: (a -> a -> a) -> [a] -> a
foldr1 f [x] = x
foldr1 f (x:y:xs) = f x (foldr1 f y:xs)

Table 5 The sum function

sum :: [num] -> num
sum xs = foldr1 (+) xs

Note: This could also have been written from scratch giving

sum [x] = x
sum (x:xs) = x + sum xs

of their types shows that they are polymorphic in nature. So generally, higher-order functions could be considered polymorphically oriented.

The features like polymorphic functions and higher-order functions encourage software reuse.

1.3 Overview of the Thesis

Because of the ever growing need to reuse existing software, more and more resources are being directed towards reuse-oriented program development. Not only are we concern with how to handle the vast quantity of reusable software components but with a way to retrieve required ones. Also meaningful software reuse is becoming quite cumbersome as the size of the repository increases, therefore automatic retrieval of suitable software components from a reusable component repository is essential to achieve such a reuse.

Many multifaceted approaches to automatic retrieval of software components have been developed and outlined[9]. However, the work done in this thesis is centered around the type based approach. According to Zand & Samadzadeh[17] six models of
software reuse, type based can be categorized as belonging to “design for reusing existing components”, but may be extended to “design for reusability” in the future. Figure 1 shows the path type based traverse in the software construction environment.

The objective of this thesis is to investigate the use of a structured software base for reuse-oriented program development. In particular, it focuses on the first two stages of reuse-oriented program development. That is, storage and locate/retrieve. It implements a reusability software based prototype system, WISER (Windsor Software Base for Reuse), developed with this software base design enforced and geared specifically for functional programming environment. This system provides a user-friendly interface and a graphical browser which allow the user to navigate through the software base. The work covered in this thesis was not concerned only with finding exact matches but various forms of relax matches as well.

1.3.1 Motivation

As technology increases and the world becomes more automated the demand for
software increases, which means more and more reusable software components are being produced. Hence, the reusable software components are growing. We are now face with the dilemma of storing these components so as to allow easy retrieval for reuse.

The traditional approach to reuse have neglected to focus much attention on this increase of components, by the structuring of their software repository. They have basically been relying on sequential searches to find suitable candidates. This leads to the problem of having to search the repository many times before retrieving suitable candidates for reuse, as in the case of relax matches. Furthermore, it requires a great deal of human interaction by means of reformulation of query if no success was attained at first.

Finally, there is a need for the user to be able to browse through the repository of software components.

1.3.2 Our Approach: A Structured Repository

Our solution proposed is divided into three parts. The first deals with structuring the repository of software components based upon the reusability relations that exist among function types. The polymorphic and extra-argument relations that exist among functions were the ones considered and used during this phase. Polymorphic relation was employed because it allows the user to find more general and/or specific matches (part of the relax match), that could mimic the required function's behavior by slight or no modification at all. The extra-argument relation was used so as to obtain functions that could be made to simulate the required function's behavior by freezing an appropriate number of arguments (the remainder of the relax match). It is mainly interested in the retrieval of higher-order functions.

The next part of the solution deals with the user using the type of the required function as a search key with which to navigate the structured software base and retrieve suitable
candidates. By using this type and the structured software base the user does not have
to reformulate his/her query if an exact match is not found.

Finally, a prototype reusability system was developed with this software base design
enforced and geared specifically for functional programming environment. The software
base consists of Miranda\(^2\) functions. This system provides a user-friendly interface with
minimum user intervention as compared to previous type based systems. Motif was used
as a tool to aid in the construction of the interface, while C was used as an implementation
language. A graphical browser is added so as to allow the user to navigate through the
software base.

Formal specification would have been ideal for retrieval/location of reusable compo-
nents, however because it is generally undecidable giving rise to unimplementable, we
have decided to use a simple form of specification in terms of types. By using the type
of the required function as a search key into the software library one is able to obtain
possible candidate functions for reuse. As an example of how this is performed, suppose
we were to retrieve a function \( f \) that is defined as :

\[
f \ g \ [a_1, \ldots, a_n] = [ga_1, \ldots, ga_n]
\]

then its type \((a \rightarrow b) \rightarrow [a] \rightarrow [b]\) can be used as our search key into the library.

With this idea of using types as search keys suppose we want to find a function that
returns the first element from a list of elements. Using the software repository shown
in Figure 2, we can use \([a] \rightarrow a\) (\(a\) was just chosen at random, any type variable could
have been used, such as \(v\)) as our search key, which according to our system should
return \(hd\) since an exact match was found. Now what about a function that concatenates
a number to the beginning of a list of numbers. The search key for such a function could

\(^2\) Miranda is a trade-mark of Research Software Ltd.
be \((\text{num} \to [\text{num}] \to [\text{num}])\) or \([\text{num}] \to \text{num} \to [\text{num}]\). since it is impossible for
the user to know the order of the arguments in the stored types. This thesis do take the
argument order into consideration, thereby allowing the system to retrieve \(\text{cons} :: a \to [a] \to [a]\). as in this case of a no exact match scenario (chapter 2 explains how this is
possible). Taking the argument order into consideration means keeping the result type
fixed while all possible combinations of the arguments are checked for an exact match.

This thesis also deals only with the curried\(^3\) version of functions. By curried we mean
the function has type \(a \to b \to c \to \ldots \to n-1 \to n\) and not the uncurried version \((a, \ b, \ c, \ldots, \ n-1) \to n\). So for the above example \([\text{num}], \text{num} \to [\text{num}]\) and \([\text{num}, \text{num}] \to [\text{num}]\) would not be appropriate search keys for the concatenation function.

Types were chosen as search keys to aid in the retrieval/location phase of reuse-
oriented program development for the following reasons:

1. Types are inherit in functions and as such does not need to be manufactured or
   derived.

---

\(^3\) Named after the American logician H. B. Curry
2. A number of types can be eliminated very easily when searching thereby constraining the search.

3. Eliminate conflicts when combining components as in the case of functional composition. If we have functions corresponding to the three types

   i. \( a \rightarrow [a] \rightarrow [a] \)

   ii. \([a] \rightarrow \text{bool}\)

   iii. \(\text{num} \rightarrow \text{bool}\)

   and requires a function of type \(a \rightarrow [a] \rightarrow \text{bool}\). Now if this type does not exists in the software repository, then by composing the first two types we would be able to obtain the required type and ultimately a possible function composition. Other composition of these types would not be possible and as such can be readily ruled out. Unfortunately this thesis does not deal with the retrieval of components based upon the automatic composition of a number of types (functions).

4. Types are very important for reliable software since everything concerning types are done at compile time, and as such causes a high proportion of programming errors to be found at an early stage.

1.3.3 Thesis Statements

This thesis defends the following statements:

1. Organizing the components of a software repository based upon the reusability relations that exist among them, so as to support for more effective reuse in the large.

2. Implementation of a prototype software base, specifically for functional programming environment (Miranda).
3. Testing of system by means of applications to determine how well it supports reuse-based program development.

1.3.4 Organization of Thesis

This thesis is organized into five chapters. Chapter 1 gives a brief discussion on the various approaches to automatic software retrieval, and an overview of the thesis.

In Chapter 2 we explain in detail the structuring of the software base which entail the various relations that exist between types. Also the record’s structure is explained.

Chapter 3 describes the design and implementation of the prototype system called WISER.

In Chapter 4 we describe WISER’s user-friendly interface and presents some applications of WISER in developing Miranda programs through reuse. Finally Chapter 5 highlights the conclusions along with some future work.

Added to the chapters are three appendices: Appendix A containing a sample of the structured software base, Appendix B providing the source code listing for WISER and Appendix C its user manual.
Chapter 2  Software Base : A Structured Approach to Repository

The design of a software base is the key concept in the storage phase of reuse-oriented program development. The structuring technique proposed and implemented in this thesis is unique in that it uses both polymorphic and extra argument relations among components. Furthermore, a number of files are used as a medium for storage of our components. With such a software base the retrieval of suitable functions could be obtained simply by navigating the graph.

This chapter addresses the first phase of reuse-oriented program development by focusing on the various aspects of a structured software base. It explains the many relax matches possible and also the relations used in its formation.

2.1 Functional Generalization

One of the most promising concepts of reusing existing software components, surrounds the idea of functional generalization. This idea simply means using functions that exhibit all the behaviors of the required function, thereby simulating near misses.

Functional generalization as a whole can be partitioned into two distinct categories, the first being general functions while the second is functions with more arguments.

2.1.1 General Functions

A function $F_1 :: \alpha$, is said to be more general than a function $F_2 :: \beta$, if $\beta$ is an instance of $\alpha$ (i.e $\beta \prec \alpha$). The general function, $F_1$ could perform the same operations as the function $F_2$ plus more. For example if we have:

$$F_1 :: [\text{int}] \rightarrow [[\text{int}]] \rightarrow [[[\text{int}], \text{int}]] \quad \text{and}$$

$$F_2 :: [a] \rightarrow [\text{int}] \rightarrow [(a, \text{int})]$$
then the first attempt at determining whether one type is an instance of the other might seem impossible. This is because we are trying to unify \([a]\) with \([\text{int}]\), \([\text{int}]\) with \([[[\text{int}]\]]\) and \([([a, \text{int}])\) with \([[[[\text{int}].\text{int}]]]\), which returns neither being an instance of each other. However, we take the arguments order into consideration. Therefore by rearranging the arguments order, all possible instances should be checked until a finite answer is obtained or all possible combinations are exhausted. By checking these instances as shown in Table 6, the only one that proved conclusive was D with the substitution \([\text{int}]\) for \(a\) in \(F_2\). Thus concluding that \(F_2\) is more general than \(F_1\) and in this case might simulate the behavior of \(F_1\).

In the context of software retrieval, scenarios will occur making the need for general functions a reality. Three such possible scenarios are listed below:

1. Suppose no exact match is found in our software repository. Then a more general function could be used if one exists.

2. If an exact match is found yielding functions that are of no significance to the user, then more general function should be pursued.

3. If the user cannot formulate the most general type for his/her required function, but can however give an example of what is needed. Then, if the desired function does exists in the software repository, it would be more general than the actual querying one. Therefore by retrieving the more general functions, the user will be able to find his/her desired function.
**Query**

\[ \text{[num]} \rightarrow \text{[num]} \]

**S/W Library**

As an example, if a user needs a function that acts on a list of numbers by returning the same list of numbers, but in the reversed order. That is, if the function is called \( \text{reverse} \) then \( \text{reverse} \{1, 2, 3, 4\} = \{4, 3, 2, 1\} \). By analyzing this function we see that reverse is of type \( \text{[num]} \rightarrow \text{[num]} \). Using this type as a search key into our software repository will fail in yielding an exact match (see Figure 3). However, if the avenue of more general functions were pursued, then we will obtain \( \text{tl} :: [a] \rightarrow [a] \) and \( \text{revList} :: [a] \rightarrow [a] \) (by substituting \( \text{num} \) for \( a \)). Finally, by inspecting the source code for both \( \text{tl} \) and \( \text{revList} \), the user could decide which if any of these functions is appropriate. Functions more general than the desired ones could sometimes be used to mimic the desired function’s behavior.

2.1.2 Functions with more Arguments

Sometimes it is possible to use functions that are obtained from under the freezing argument principle to duplicate your desired function’s behavior. Under this principle the result of the function is kept fix while freezing is done on an appropriate number of arguments. Thus, producing a type that is identical to the type of your desired function.

For example if the user requires a function that sorts a list of numbers. Then by analyzing that functions’ needs, a possible query type might be \( \text{[num]} \rightarrow \text{[num]} \). Also,
suppose a subset of our software repository is as shown in figure 4. Now by using this type as a search key into this software repository will fail in yielding an exact match. However, if we were to retrieve functions based on the freezing argument principle, a number of possible alternative solutions might be available, as in the case of numSort. If the first argument, \((\text{num} \rightarrow \text{num} \rightarrow \text{bool})\), of numSort was kept frozen, the resulting type would be \([\text{num}] \rightarrow [\text{num}]\), which is identical to the query type. Finally, by examining the source code for the numSort function the user can then decide whether this function could be used instead.

As a final example of functional generalization, “suppose a user needs to convert a list of integers to a list of boolean values, where each boolean corresponds to whether or not the corresponding integer is positive”[18] (i.e. \(f[1, 2, -1, 4, -2, -3] = [T, T, F, T, F, F]\)). Assuming that the user wants to retrieve this function from our software repository (same as the one in Figure 4) and integrate it into his program. As with any programming languages there are two kinds of programmers, the inexperience and the experience one. So, if the inexperience programmer requires such a program it is highly possible that the query type would be \([\text{num}] \rightarrow [\text{bool}]\), whereas the experience one might use \((\text{num} \rightarrow \ldots\))
bool) → [num] → [bool] as their search keys. We examine both cases below.

Using the query type (num → bool) → [num] → [bool] as a search key into the software repository, would fail in yielding an exact match. But the more general type (a → b) → [a] → [b] would be retrieved and its associated function, map, examined to see if it is deemed appropriate. Finally, by using [num] → [bool] as a search key, would also fail in yielding an exact match. However, by incorporating both freezing argument principle and more general function into the search we would be able to also obtain map :: (a → b) → [a] → [b]. Retrieval of this form will be dealt with in more details, in the latter part of this chapter. Thus it is possible to obtain the same function through different query types.

2.2 Functional Specialization

A function, \( F_1 :: \alpha \), is said to be more specific than a function, \( F_2 :: \beta \), if \( \alpha \) is an instance of \( \beta \) (i.e. \( \alpha < \beta \)). This is basically saying that the function \( F_1 \) can probable be used to replace the function \( F_2 \) by generalizing \( F_1 \)'s source code. This is based on the assumption that the source code is written reasonable well.

A user might query with a general type that does not form an exact match with any of the types in the software repository. There may however be a useful function in the repository whose type is more specific, but whose source code could be easily generalized to be useful to the querier. For an example of such a scenario, suppose the user requires a general function to sort lists. Using our software library (same as in Figure 4), the user could query with the possible general type \( (a \rightarrow a \rightarrow bool) \rightarrow [a] \rightarrow [a] \). Unfortunately our library does not contain any function with that corresponding type, but under specialized match would return numSort :: (num → num → bool) →
Figure 5 Sample polymorphic relation between types.

\[ a \rightarrow b \]

\[ \text{[num]} \rightarrow \text{[num]} \]

Now assuming \textit{numSort} is written intelligently enough, the user should have no trouble in modifying it to sort arbitrary entities.

2.3 Structuring of Software Components Repository

The structuring of the software base was accomplished based on the functional relationships explained earlier. That is, making use of the general, freezing argument and specific relations that exists among functions. These three relationships were decomposed into two separate relations. The first, called polymorphic relation, uses the conjunction of both general and specific function relationships. While the second relied on the freezing argument function relationship and is called extra-argument relation.

2.3.1 Polymorphic Relation

By taking advantage of the general and specific relationships that exists among functions, a graph was constructed so as to allow for easy navigation and ultimately, direct access to general and specific functions. Figure 5 shows a sample of such a graph based upon this relation. An examination of this figure shows the use of double directional links between types, this is because of the general and specific relation that
exists. For example, we have

\[(a) \rightarrow b\]

\[(a) \rightarrow num\]

which implies that \(a \rightarrow b\) is more general than \(a \rightarrow num\) (i.e. \(a \rightarrow [b]\) is a parent of \(a \rightarrow num\)) and \(a \rightarrow num\) is more specific than \(a \rightarrow b\) (i.e. \(a \rightarrow num\) is a child of \(a \rightarrow b\)). A type can have many immediate general types and also many immediate specific types as in the case of \(a \rightarrow [a]\) (See Figure 6). Furthermore, a root type has no parent and is the most general of all types in the graph. A leaf type has no children, thus it is the most specific type, but not for all types in the graph.

By navigating the graph in Figure 5 the user can access all functions that are more general than the query type. By “all functions more general”, we mean not just the immediate ones but also those that are not immediate, as in the case of ancestors. By the same token, they can access all the more specific types (predecessors) which includes grandchildren.

Lets consider the situation where the querier uses \(a \rightarrow [a]\) as a search key. By navigating this graph we encounter an exact match (the way the search is actually done is covered in details in chapter 3). If this type does not contain a function that is usable,
the querier then has the option of using more general or specific functions. In this case the more general types are \((a \rightarrow b), (a \rightarrow [b])\) and \((a \rightarrow b)\) while the specific ones are \(([a]) \rightarrow ([a]), ([char]) \rightarrow ([char]), ([num]) \rightarrow ([num])\) and \(([num]) \rightarrow ([num])\). So, if none of these proves fruitful more general and/or more specific types, of these can be explored. In the case of not finding an exact match, the type will be inserted automatically (covered in details in chapter 3) and the querier continues to search for general and/or specific functions.

2.3.2 Extra-Argument Relation

We have showed earlier how functions whose types contains more arguments than the queried type can be of immense importance to the querier. By structuring the types in the software repository with this freezing argument relation in mind, would provide direct access and ultimately sustain useful retrieval of software components. A sample of types from our software repository with this relation enforced is shown in Figure 7.

This kind of structure uses a one directional link originating from types of less arguments and converging on those containing more arguments in a meaningful way. Traversing these links will help the querier in retrieving possible useful functions. Not only will they be able to use components with one argument more than the queried
one, but components with more than one extra arguments, by performing the appropriate freezing. For example, if the user queries with the type \( a \rightarrow a \) and no useful function can be found by performing exact, general and specific matching, then the path of extra argument should be pursued. For this query and using Figure 7, the possible extra argument types are \((a \rightarrow [b] \rightarrow a)\), \((a \rightarrow a \rightarrow a)\), \(((a \rightarrow \text{bool}) \rightarrow a \rightarrow a)\), \(((a \rightarrow b \rightarrow a) \rightarrow a \rightarrow [b] \rightarrow a)\), \(((a \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b)\) and \(((a \rightarrow \text{bool}) \rightarrow (a \rightarrow a) \rightarrow a \rightarrow a)\). Furthermore, general and/or specific matches under these extra argument types can also be pursued based upon the polymorphic relation.

2.3.3 Sample Structured Software Base

The software repository was structured by integrating both polymorphic and extra-argument relations among all types in the library, thereby acquiring the best of both worlds. With these relations in place the user now has multiple options available to him/her in the field of automatic software retrieval for reuse. Figure 8 and 9 shows very small samples of the interconnection of types based on these relations. Very small samples were used so as to avoid confusion and congestion thus, enabling the portrayal of clarity.

Using the sample repository of Figure 8, suppose a user queries with type \( \text{bool} \rightarrow [\text{bool}] \rightarrow \text{bool} \) which yields an exact match. But, if unfortunately no useful function of
that type could be found, then the user has the opportunity of exploring more general, more specific and extra-argument functions. In this case no immediate specific type or immediate extra-argument types exists. Thus the only immediate more general type, \( a \rightarrow [b] \rightarrow bool \), is explored. If this more general type proves hopeless also, then its specific and general types can be explored. However, the querier can also take advantage of the other possible solution available, as in the form of the extra-argument link of type \( a \rightarrow [b] \rightarrow bool \), which is \( (a \rightarrow b \rightarrow bool) \rightarrow a \rightarrow [b] \rightarrow bool \). The cycle can then be repeated based on the new types retrieved as the situation dictates.

In order to reduce the number of links in the software repository while still maintaining integrity, a technique was implemented during the construction of links formed by the extra-argument relation. This technique is best explained by analyzing Figure 10. As you can see from this figure the type \( a \rightarrow bool \) has two links based on the extra-argument relation to \( a \rightarrow a \rightarrow bool \) and \( a \rightarrow [a] \rightarrow bool \). However, according to this relation \( a \rightarrow bool \) should also have a link to \( a \rightarrow [b] \rightarrow bool \). But because \( a \rightarrow [b] \rightarrow bool \) is a more general than type \( a \rightarrow [a] \rightarrow bool \), all we have to do is follow the polymorphic relation based on more general types and obtain such from \( a \rightarrow [a] \rightarrow bool \). Therefore this technique allows for the maintenance of links corresponding to the most specific types in
Table 7 Record structure

\[ \Sigma \{ F_\Sigma \} \{ G_\Sigma \} \{ S_\Sigma \} \{ H_\Sigma \} \]

\( \Sigma \) represents the type
\( F_\Sigma \) represent all the functions with type \( \Sigma \)
\( G_\Sigma \) represents the offsets of all the immediate most general type(s) of \( \Sigma \)
\( S_\Sigma \) represents the offsets of all the immediate most specific type(s) of \( \Sigma \)
\( H_\Sigma \) represents the offsets of all the immediate extra-argument type(s) of \( \Sigma \)

the set of related types. For a more practical explanation consider the set of related types, \{ \{a \rightarrow a \rightarrow bool\}, \{a \rightarrow [a] \rightarrow bool, a \rightarrow [b] \rightarrow bool\} \}, corresponding to type \( a \rightarrow bool \). Using the most specific type from this set will give us the links shown in Figure 10.

So ultimately the user controls the direction and the extent of the search performed.

### 2.4 Structure of a Record

For the structure of the software base to be made possible, records containing a number of fields were used to imbed types along with a number of useful information. These records were then stored in various files depending upon the number of arguments their type contained. Simply put all records whose type contains zero argument were placed in one file, those with one argument were placed in another, and so on. Table 7 shows the format of such a record.
To gain an understanding of the fields in the record more fully consider the sample record with its associated fields:

\[(\{\text{a} \to \text{a}\}, \{\text{hd, last, limit, max, min, middle}\}, \{250\}, \{5250, 8250, 10000\}, \{5750, 8500\})\]

We can deduce from this record that the functions, \text{hd, last, limit, max, min, middle}, has type \(\text{\{a} \to \text{a}\), also at offset 250 you will find a immediate record whose type is more general than \(\text{\{a} \to \text{a}\). At offsets 5250, 8250 and 10000 immediate records whose type is more specific than \(\text{\{a} \to \text{a}\) exists. Finally, at offsets 5750 and 8500 in the file whose records contains type of two arguments, you will find immediate records that adhere to the freezing argument principle.

A sample of the software repository is given in Appendix A.
Chapter 3 WISER: A Reusable Software Base System

In keeping with the concept of not reinventing technology, a prototype reusable software base system called WISER (WIndsor SoftwarE base for Reuse) was designed and developed. It is an interactive software reuse system that is directed at the first two stages of reuse-oriented program development. The concerns of WISER are centered around the construction of functional programs like Miranda programs and as such maintains an evolving structured software repository of primitive Miranda components.

The purpose of most automatic software retrieval system is to support reuse-oriented program development and as such WISER is no exception. WISER offers a wide range of features to users, thus enabling them to perform a number of useful operations. The concerns of this system are centered around the concept of being able to browse the structure of the repository alone with the retrieval, insertion and deletion of functions, with more emphasis directed towards the insertion. This is because situation will arise making retrieval dependent upon insertion. Furthermore, it is of significant importance for the system to be able to handle retrieval of software components from the software repository, conforming to its design.

WISER can be integrated into the source code phase of traditional software development life cycle[2]. So that when writing source code for Miranda applications the software developers could use WISER to aid in the reuse of Miranda functions by interacting with WISER’s software repository.

All of the features entailed by this system relied on the process of determining whether one type is an instance of another and as such a modification of Robinson unification algorithm[14, 5] was incorporated into the instance detection procedure.
This chapter covers the design of such a system and explains how the second phase of reuse-oriented program development is accomplished. The evolving nature of the software base is also addressed. Focus is placed on the documentation of the location/retrieval, deletion and insertion mechanisms used in the interaction with the software repository.

3.1 System Design

During the design phase of WISER a number of ideas were implemented with the hope of producing a system that will ultimately help the users in their program development. Figure 11 shows an overview of WISER’s design.

A way of constructing a structured software repository of Miranda functions suitable for Miranda program development was at the pinnacle of WISER’s design. Furthermore a way of managing this library of reusable components was needed, since it will ultimately aid in the construction of the structured software repository. The library management focuses on the three design principles given below:

1. It was figured that the insertion of components (Miranda functions) and their associated types should be done in an automated way by the system and as such expounded on the need for the design of an insertion tool.

2. Since the system relies on the concept of maintaining a growing software base, the insertion of types without any associated functions must also be identify. This is done so that future attempts to insert a component into the software base might find a type that performs an exact match, thus allowing for the simple insertion of that function without having to change any links.

3. The system design has to make allowance for the replacement of functions by more efficient ones, as time progresses. To make this possible the system needs to remove
(delete) old less efficient functions and insert the new more efficient ones in their place.

Once the concept of the software repository was established the concerns were then directed towards the automatic retrieval process. The design of such a retrieval tool
should not only allow for retrieval based on exact matches but various forms of relax matches as in more general, more specific and extra-argument functions. Also in the case of not finding an exact match, allowance must be made to insert it unbeknownst to the user, thereby allowing our software repository to grow. Because WISER maintains a structured software base a way and means of browsing it must be made possible. Finally the system is geared towards users by relying on their queries, making the obligation for a user-friendly interface to the library a must.

3.2 Code Documentation

3.2.1 Insertion of components

Due to the complexity of the insertion process a number of conditions must be checked and satisfied, so that the insertion of a component into the software base can be expedited without any errors. Figure 12 shows the basic structure of the insertion process. According to this figure the insertion starts with the user supplying the required information, which is in the form of the function's type and corresponding name. A verification is then submitted by the system in response to the query so that corrections can be made in case of errors in the query. Next that function is then inserted into the repository in accordance with its type. If there does not exists a type in the software repository that corresponds to the query type, then a new record containing the query type is created and inserted. Furthermore types formed by all possible combinations of arguments (using the original type) based on freezing are inserted automatically, if need be, but without any function names attached to them. If an exact match with the query type was found then the function name is simply inserted. Figure 12 also shows calls to a number of functions that are used to satisfy a number of conditions, thereby making the insertion possible. The use of these functions are explained later on.
The basic test involved in such an operation is deciding whether one type is an instance of another and will be used extensively throughout this procedure. We know
that if a type $\alpha$ is an instance of a type $\beta$ (i.e. $\alpha \prec \beta$) then $\alpha$ is said to be more specific than $\beta$ and vice-versa. So using this information we can proceed to insert components into the software repository. But before any insertion can take place, the root of each file must be set. The root is a record containing the most general type of all record’s type in that particular file. Furthermore they are linked together based upon the extra-argument relation. Therefore after their insertion we should have a structure that looks like Figure 13.

The simplest way to understand the insertion operation is by following the insertion of various components thereby encompassing all of the techniques used. Also remember the insertion is in the form of records with fields as discussed in chapter 2, so when types are mention they refer to the corresponding record’s type. The insertion of the function name into the record done by updating the relevant field. Insertion consists of determining the insertion record’s children, parent and extra-argument links as well as updating the various links for certain records, based on their type, in the software repository. We will focus on all of the conditions necessary for insertion based on polymorphic relations first and then on those based on extra-argument relations.

**Insertion via Polymorphic Relation**

Assuming that only the root node exists in the various files we will then proceed to
insert the type \([a] \rightarrow b\). After this insertion the structure of the one argument file is as shown in Figure 14. This is because:

- \([a] \rightarrow b \prec (a \rightarrow b)\).

Next we insert \(a \rightarrow [b]\) yielding Figure 15. This is because:

- \((a \rightarrow [b]) \prec (a \rightarrow b)\).
- \((a \rightarrow [b]) \not\prec ([a] \rightarrow b)\) and
- \((a \rightarrow [b]) \not\prec (a \rightarrow [b])\).

Continuing along we then insert \([[a]] \rightarrow b\) giving Figure 16. This is because:

- \([[a]] \rightarrow b \prec (a \rightarrow b)\).
- \([[a]] \rightarrow b \prec ([a] \rightarrow b)\).
- \([[a]] \rightarrow b \not\prec (a \rightarrow [b])\) and
- \((a \rightarrow [b]) \not\prec ([[a]] \rightarrow b)\).
The insertion seems to be based upon instances alone and once an instance was found we link it and that was it. However this is partially correct as shown in Figure 17, according to the insertion of \([a] \rightarrow [b]\). The relevant links were formed because:

- \(([[a]] \rightarrow [b]) < (a \rightarrow b)\).
- \(([[a]] \rightarrow [b]) < ([a] \rightarrow b)\).
- \(([[a]] \rightarrow [b]) < ([a] \rightarrow b)\) and
- \(([[a]] \rightarrow [b]) < (a \rightarrow [b])\).

So far we have shown that although a parent was found, does not mean that no more parents exists as in the case of Figure 17. Therefore we must continue searching to determining whether or not it has more parents. Also searches were done from left to right and top to bottom recursively. Part of the function insert tests for these conditions mentioned above.

Lets now look at the insertion of \(a \rightarrow a\) into the structure of Figure 18. According to this figure we have:

- \((a \rightarrow a) < (a \rightarrow b)\),
- \((a \rightarrow a) \not< ([a] \rightarrow b)\),
- \(([a] \rightarrow b) \not< (a \rightarrow a)\),
- \((a \rightarrow a) \not< (a \rightarrow [b])\) and
Figure 18 One-argument structure before insertion of $a \rightarrow a$

$\begin{align*}
\triangle
\text{a} \rightarrow \text{b} \\
\text{v}
\end{align*}$

$\begin{align*}
\text{[a]} \rightarrow \text{b} \\
\text{v}
\end{align*}$

$\begin{align*}
\text{[[a]]} \rightarrow \text{b} \\
\text{v}
\end{align*}$

Figure 19 Incorrect insertion of $a \rightarrow a$ into one-argument structure

$\begin{align*}
\triangle
\text{a} \rightarrow \text{b} \\
\text{v}
\end{align*}$

$\begin{align*}
\text{[a]} \rightarrow \text{b} \\
\text{v}
\end{align*}$

$\begin{align*}
\text{[[a]]} \rightarrow \text{b} \\
\text{v}
\end{align*}$

$\begin{align*}
\text{[[a]]} \rightarrow \text{[[a]]} \\
\text{v}
\end{align*}$

- $(a \rightarrow [b]) \neq (a \rightarrow b)$.

So based upon the above checks the new structure should correspond to Figure 19. However, this is not entirely correct since $([[a]] \rightarrow [[a]]) \neq (a \rightarrow a)$ and $[\text{num}] \rightarrow [\text{num}] \neq (a \rightarrow a)$. That's why we need to go back and check almost each type individually to see if it is an instance of the inserted one. Thereby allowing many children. Furthermore if a type is found to be an instance of the inserted one it is further checked to ensure that its link does not exists already and also that it is not accessible from the inserted type through any of the inserted type specific links. Figure 20 shows the correct structure for
the insertion of $a \rightarrow a$. Therefore once an instance is found none of its instances needs to be analyze since they too will be instances of the inserted type, although not immediate instances. The function $insert_{\text{final}}$ was used in testing for this condition.

We will now look at when there is a need to change a parent’s child. Consider the insertion of type $[a] \rightarrow [a]$ into the structure of Figure 20, producing the resulting structure as shown in Figure 21. From this graph we observe that:

- $([a] \rightarrow [a]) \prec (a \rightarrow b)$,
- $([a] \rightarrow [a]) \prec ([a] \rightarrow b)$ and
• \([\text{a} \rightarrow \text{a}] \neq ([\text{a}]) \rightarrow \text{b}]\).

Which shows that type \([\text{a}] \rightarrow \text{a}\) is a child of type \([\text{a} \rightarrow \text{b}]\). Continuing along we have:

• \(([\text{a}] \rightarrow \text{a}) \prec (\text{a} \rightarrow \text{b})\).
• \(([\text{a}] \rightarrow \text{a}) \neq ([\text{a}]) \rightarrow ([\text{a}])\).
• \(([\text{a}] \rightarrow \text{a}) \neq ([\text{num}] \rightarrow [\text{num}])\) and
• \(([\text{num}] \rightarrow [\text{num}]) \prec ([\text{a}] \rightarrow [\text{a}])\).

Which shows that type \([\text{a}] \rightarrow \text{a}\) is a child of type \(\text{a} \rightarrow \text{b}\) and also \([\text{a}] \rightarrow \text{a}\) has a child of type \([\text{num}] \rightarrow [\text{num}]\). Continuing along we have:

• \(([\text{a}] \rightarrow [\text{a}]) \prec (\text{a} \rightarrow \text{a})\).
• \(([\text{a}] \rightarrow [\text{a}]) \neq ([\text{num}] \rightarrow [\text{num}])\).
• \(([\text{num}] \rightarrow [\text{num}]) \prec ([\text{a}] \rightarrow [\text{a}])\).
• \(([\text{a}] \rightarrow [\text{a}]) \neq ([\text{a}]) \rightarrow ([\text{a}])\) and
• \(([\text{a}]) \rightarrow [([\text{a}])] \prec ([\text{a}] \rightarrow [\text{a}])\).

Which shows that type \([\text{a}] \rightarrow [\text{a}]\) is a child of type \(\text{a} \rightarrow \text{a}\), also type \([\text{a}] \rightarrow [\text{a}]\) has children of types \([\text{num}] \rightarrow [\text{num}]\) (which exists already and so is ignored) and \([([\text{a}])] \rightarrow [([\text{a}])]\). In this example \([\text{a}] \rightarrow [\text{a}]\) took all the children of type \(\text{a} \rightarrow \text{a}\), but in general this is not always true since it did not take all the children of types \([\text{a}] \rightarrow \text{b}\) and \(\text{a} \rightarrow \text{b}\). The reason for it not taking all of the children is that it is possible for some of the children not to be instances of it. The function \text{insert} also test for the child changing condition.

Finally there is the condition of finding an exact match of types and when such occurs the function name is simply inserted into the record if it does not exists already. The function \text{insert} is also used here.

Insertion of a function and its associated type into the software repository is not straight forward and as such should be treated with utmost scrutiny. All of the conditions
Table 8 Algorithm for insertion of all possible types based on argument combinations

```plaintext
Function argument_comb(num_of_arg, type)
    if insertion(num_of_arg, type) produces an exact type match then return
    for i = 1 . . num_of_arg
        newType := new type formed by freezing argument i
        argument_comb(num_of_arg -1, newType)
    end
end
```

described above was tested for, thus sometimes taking the search to all of the leaf nodes.

The insertion was best accomplished by using recursion. As mentioned earlier insertion of all possible types formed by all possible combinations of arguments based on freezing are also inserted. The algorithm in Table 8 shows how this is done. A practical example is given in the section on insertion via extra-argument relation below.

**Insertion via Extra-Argument Relation**

The binding of types with extra-argument links is done concurrently with the insertion of types, based on all possible combinations of arguments by freezing. So let's go back to the insertion of a function and its corresponding type. When this record is inserted, the field corresponding to the offset of extra-argument links must also be updated. To gain a more in-depth and practical outlook, we will consider the insertion of a number of functions based on both polymorphic and extra-argument relation, with emphasis directed towards the latter. Furthermore we will insert the associated type’s function name so as to show that not every type has a function associated with it, but could somewhere in the future depending upon other insertions. Assume that initially the files contain only their root type.

The insertion of a record using its type field will be done according to the ripple effect, that is, once a type is inserted we then proceed to insert other types formed by
freezing varying number of argument. So lets analyze the insertion of \((=) :: a \rightarrow a \rightarrow bool\) (see Figure 22). Following the principles of insertion we first insert \((=) :: a \rightarrow a \rightarrow bool\). If an exact match was found we do not have to worry about the ripple effect. If an exact match was not found we insert the corresponding record into the file with two argument as in this case and set the offset of extra-argument links to null. Next we freeze the first argument resulting in a new type \(a \rightarrow bool\), which is then inserted into the one argument file but without assigning any new functions to it. If an exact match was found we stop the ripple effect, update its extra-argument offset links and start back from the original type freeze another argument. However an exact match was not found so we assign the offset of the record with type \(a \rightarrow a \rightarrow bool\) to its extra-argument link field. Freezing is then applied to another argument resulting in \(bool\) which is then inserted into the zero argument file with the relevant updating. We then start back from the original type and freeze another argument resulting in \(a \rightarrow bool\). An insertion of this new type forms an exact match so we stop the ripple effect and freeze another argument from the original type. No more arguments remains to be frozen so we stop the process altogether. The resulting structure is as shown in Figure 22.
Table 9  Algorithm for updating a record extra-argument field when an exact type match is found

<table>
<thead>
<tr>
<th>Assume:</th>
<th>$R_n$ is the record containing type $T_n$, where $n$ is the number of arguments.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$R_{n-1}$ is the record containing type $T_{n-1}$ formed by freezing one of $T_n$ argument.</td>
</tr>
<tr>
<td></td>
<td>Also $R_{n-1}$ is the record being inserted.</td>
</tr>
</tbody>
</table>

If $T_{n-1}$ does not exist already in the $(n-1)$ argument file then

Set $R_{n-1}$ extra-argument offset field to the offset of $R_n$

else

1. if the offset of $R_n$ exists in the extra-argument field of $R_{n-1}$ then finish

2. elseif any of the types, corresponding to offset from $R_{n-1}$ extra-argument field, is an instance of $T_{n-1}$ then finish

3. elseif $T_{n-1}$ is an instance of any of the types, corresponding to offset from $R_{n-1}$ extra-argument field then remove those types and insert $R_{n-1}$ offset.

We will now focus on the insertion of $\text{mem :: } a \rightarrow [a] \rightarrow \text{bool}$ into our new structure (see Figure 23). So once again we first insert this record into the two argument file and set its extra-argument links to null. Next we freeze the first argument resulting in the new type $[a] \rightarrow \text{bool}$ which is then inserted into the one argument file but without assigning any functions to it and also updating its offset as before. Continuing alone we freeze another argument resulting in $\text{bool}$ which already exists in the zero argument file, so its extra-argument link field needs to be updated. The updating of $\text{bool}$ extra-argument links is accomplished by following the simple algorithm given in Table 9 (this algorithm is a part of that used in the function $\text{freeze_extra_arg}$ which does the entire updating of the extra-argument offset field, i.e for non-exact type match too). So now we back up to the original type and freeze a different argument giving the resulting type $a \rightarrow \text{bool}$. This type exists already, so the ripple effect is aborted and the record corresponding to type $a \rightarrow \text{bool}$ is updated according to the algorithm in Table 9. No more argument remains to be frozen so the insertion is completed. Figure 24 shows the structure after insertion.
of member :: (a → b → bool) → a → [b] → bool. This was accomplished by making use of all the principles of insertion outlined, which includes the algorithm in Table 9.

3.2.2 Retrieval of components

The underlying purpose of any software reuse system relies heavily on being able to retrieve reasonable candidate functions. Retrieval conforms to the process of navigating the graph constructed by the interconnecting links among records in the software base. Figure 25 shows the structure of the retrieval process, according to which the user submits his query in the form of the function’s type and the number of arguments contained in it. A verification is then asked for by the system in response to the query so that changes can be made if needed. Next a search is done on the software repository. This search restricts the number of types examined by focusing only on those containing the number of arguments specified by the user. The way this is done is by using the file containing the number of arguments specified. If a record is found whose type produces an exact match with the query type then the functions associated with that record are retrieved. Furthermore by traversing the various offset links associated with this record; general, specific and extra-argument functions can be retrieved. If the retrieval type does not exist in the repository it is automatically inserted in accordance with the insertion
process discussed above, and is oblivious to the user. The principle of retrieval based on insertion will aid in the retrieval of various forms of relax matches by following the newly created links of the newly inserted record.

The search technique involved in the retrieval process relies on the basic fact that, if
Table 10: Exact match retrieval algorithm

```plaintext
flag := 0
Function exact_retrieval(retrieval_type, record_offset, num_arguments)
    get all children, C₁ .. Cₙ, of record at record_offset
    for i = C₁ .. Cₙ
        get type, Tᵢ, from Cᵢ;
        type_offset := offset of Cᵢ;
        if retrieval_type < Tᵢ then
            if exact(retrieval_type, Tᵢ) then
                return function_names from Cᵢ along with type_offset
                flag := 1;
                exit for loop;
            end if
        end if
    end for
    if flag = 1
        flag := 0
        return retrieval(retrieval_type, type_offset, num_arguments)
    else
        insertion(retrieval_type, num_arguments)
        return retrieval(retrieval_type, 0, num_arguments)
    end if
end exact_retrieval
```

the query type is an instance of a type that belongs to a record in the software repository, then you need only check the descendents of that record. Table 10 outlines a function that could be used to find components that corresponds to the query type. Before calling this function a check is made with the root record of the corresponding argument file to determine if an exact type match exists, in which case it would return the functions associated with that record and also its offset. Otherwise, `exact_retrieval` is called with its parameters being instantiated to the query type, zero (0) and the number of argument(s) contained in the type, respectively. Using the record stored at `record_offset`, the function
then goes through all that record's children until one is found that either performs an exact match, in which case the components stored with it are returned along with its offset, or is more general than the query type, in which case the search is restricted to the descendants of it. Furthermore this function calls itself recursively thereby narrowing down the search further.

There are two ways in which the function, *exact_retrieval*, determines when the query type does not exists. The first is when we encounter a leaf record that is more general than the query type. The second occurs if we are at a record whose type is more general than the query type but none of its children performs an exact type match and also none are more general than the query type. When these situations occur the function calls the insertion procedure to insert the query_type, but without any function associated with it and then calls itself again.

The *exact_retrieval* function returns the offset of the record, R, that contains the type that performed the exact type match, because this will be used to find general, specific and extra-argument functions.

The process of finding more general functions is accomplished by returning all functions stored with records that are ancestors of the record, R. This is done recursively by using the general offset links. The same is done for more specific functions with the only difference being that it uses the descendents of R along with the specific offset links. Finally for extra-argument functions, the file containing types with one more argument than the query type is used and the functions associated with the offsets stored in R's extra-argument offset field are retrieved.

For a practical look at the search involved, consider the structure in Figure 26 along with the retrieval of type *[bool] → bool*. The very first thing that the search does is narrows down the search plane to components with only one arguments as shown in the
It then uses the algorithm in Table 10 to obtain the required record thereby resulting in the search path shown in Figure 26. Finally the function(s) pertaining to this type can be extracted from that record.

### 3.2.3 Deletion of components

As time progresses functions will be developed that are more efficient than their counterpart in the software repository and as such, ways and means of replacing their counterpart with the more efficient ones must exists. WISER offers a way to handle the deletion of a function as shown in the Figure 27. According to this figure the user submits his query in the form of the function's type, the number of arguments contained in it and the name of the function to be removed. A verification is then sought for by the system in response to the query so that changes can be made if needed. A search is then performed so that the record containing that type could be retrieved and the required function removed. If the function or the type does not exists in the software base then an appropriate error message is transferred to the user.
The search used in finding the record that contains the required type is similar to that used in the retrieval process (see Table 10). With the difference being that it does not insert the type when no type match is found, but rather reports an error. After removing the function name from the corresponding record field, the source code is then deleted by removing that function's file.
Chapter 4  Program Development using WISER

This chapter describes the user-friendly interface that was developed for this system using Motif. Furthermore some actual applications of WISER in the functional programming environment of Miranda are discussed. The source code for WISER is available in Appendix B and its user manual in Appendix C.

4.1 The User Interface

All interaction with WISER is done through a window-base I/O and graphical user-friendly interface, constructed with the windowing facilities of Motif. This interface was designed so as to allow uniformity among windows and consequently attract new inexperience users. A number of help menus are also provided with the hope of navigating the user successfully through the required motions necessary to formulate a query and view its results. Figure 28 shows the hierarchy of the windowing structure available to
the users. At the root of this hierarchy lies the main window that spawns a number of useful help and input dialog windows, most of which in turn has their own dialogs.

The user starts his/her interaction with the system by first confronting the main window (see Figure 29) and based upon the action invoked, dialog windows are made available. The main window can be partitioned into two parts, the first can be thought of as consisting of the main menu while the second can be considered as being textual
in nature. From the main menu the user can invoke a number of useful dialogs such as the retrieval, insertion, deletion and browse dialogs with which to enter their queries. The textual part is responsible for displaying the source code of various functions as initiated by the user. If the user decide that he/she would like to view the source code of a particular function then all they have to do is click on that function in the various dialogs. This will cause the source code for that function to be displayed in textual part of the main window. The insertion, retrieval and deletion dialogs are similar in appearance so that uniformity is maintained. We will now look at the various dialogs by focusing on actual example usage.

4.1.1 Examples of Retrieval

Suppose the we wish to obtain a function that finds the head of a list of elements, then we query WISER's software repository with type \( [a] \rightarrow a \) by using the retrieval dialog as shown in Figure 30. This query produces an exact match causing the dialog window
in Figure 31 to display its results. By examining the source code for these functions we were satisfied with the function \textit{hd} being the required one. But just for curiosity let's examine the retrieval dialog window further. From this window the user is offered a number of choices, all of which can be accomplished without having to reformulate the query. Using this dialog the user can query the system to find functions obtained by functional generalization (i.e. general and extra-argument functions) and also those that are more specific than the query type. Figure 32 shows the dialog window for functions that are more specific than those with type \textit{fa} \rightarrow a. Figure 33 shows the dialog window for those formed by the freezing argument principle, while Figure 34 shows the dialog window for those function(s) that are more general than \textit{fa} \rightarrow a.

For the next query suppose the user needs a function that acts on a list of numbers by
returning the same list of numbers, but in reverse order. The query type for this will be
\( \text{num} \rightarrow \text{num} \). Figure 35 shows the function obtained by exact match. Unfortunately this proved to be useless upon viewing its source code. So more general ones were then requested and are shown in Figure 36. Upon inspecting their source code, the function reverse stood out as being able to perform the task of the required one.

For a final query consider finding a function to sort a list of numbers. The query type for this will be \( \text{num} \rightarrow \text{num} \), yielding the same function as in Figure 35 for exact match and Figure 36 for more general match. From the more general functions dialog we observe that we could use the sort function. However lets see what happens if we pursue alternative possibilities. When we tried to find more specific functions we encountered a dead end, since none exists and as such functions with extra arguments were then sought. Figure 37 showed the resulting functions obtained from the freezing argument principle. Upon examining the source code for these functions we observe that numsort satisfy the criteria of the required function.
Figure 37 Extra-argument functions

If an exact match is not found then a relevant message appears with the options to pursue general, specific and extra-argument functions. Also appropriate messages are displayed if any of these does not exist.

4.1.2 Example of Insertion

Let's now focus on the insertion of the function \texttt{concat}, by using the insertion dialog shown in Figure 38. This function has two arguments with corresponding types \texttt{a} and \texttt{[a]} and result type \texttt{[a]}. So by applying this information to Figure 38 will result in a successful insertion if the function does not exists already. Otherwise the window in Figure 39 appears.
4.1.3 Example of Deletion

If the user wants to delete the \texttt{concat :: a \rightarrow [a] \rightarrow [a]} function then all he/she has to do is enter the relevant information into the deletion dialog as shown in Figure 40. That function will then be deleted provided that its type along with itself exists otherwise Figure 41 and 42 shows the relevant error messages.
4.1.4 Example of Browser

Finally as mentioned before a browser is also implemented to allow the user not just to view the structure of the software base but also to aid in finding possible candidate functions. This browser (see Figure 43) displays all the parents, children and extra-argument type of a particular type. It also displays all the functions associates with that particular type if any exists. Using this browser the user can control which part of the structure to browse by changing the number of arguments the type contained along with moving up, down and across the structure.

The browser can be used as an aid in finding possible candidate functions when the retrieval query failed in finding any usable function, which can be attributed to the fact that types will exists without any function(s) being attached to them and as such the retrieval query would not show these types although they could be more general or more specific than the query type. So the browser could be used to traverse the links from
the retrieval query type leading to these types and subsequent checks can be made to their specific, general and extra-argument types to see if any function exists and if so the system can then query with the new functionless types and repeat the cycle if necessary.

4.2 Applications of WISER

Figure 44 The prime numbers building block diagram

A sequential list of numbers from 2 to n

\[ \begin{array}{c}
2 \\
\vdots \\
n
\end{array} \quad f \quad \rightarrow \quad g \quad \rightarrow \quad \text{List of prime numbers}
\]

Generates a sequential list of numbers from 2 to n

Removes all prime numbers from a list of numbers

4.2.1 Prime numbers

This first application focuses on generating a list of prime numbers up to a specific integer \( n \), according to the method described by the Greek mathematician Erathosthenes. His method consists of four parts as listed below:

P1. Generate a list of numbers \( 2, 3, 4, \ldots, n \);

P2. Mark the first element \( e \) in the list as prime;

P3. Delete all elements in the list that is divisible by \( e \);

P4. Repeat this process (from P2.) using the resulting sequence until the resulting sequence is empty.

The building block diagram of this problem is as shown in Figure 44. However, this is the initial design and as such is not invincible to changes.

We will solve this problem by concentrating on each of the parts (P1 .. P4) individually, starting with P1. This part requires us to generate a sequential list of
numbers from 2 to n. So based on this we need a function that takes in two numbers as arguments and produces a list of numbers as its result. The most likely type of such a function would be \( \text{num} \rightarrow \text{num} \rightarrow \text{[num]} \) and was therefore used as the query type. After successfully querying WISER, the system returned two functions with this type (exact match) as listed below:

1. \( \text{ComDiv} :: \text{num} \rightarrow \text{num} \rightarrow \text{[num]} \)
2. \( \text{GenList} :: \text{num} \rightarrow \text{num} \rightarrow \text{[num]} \)

Upon examination of \( \text{GenList} \)'s source code, we knew our search for the required function pertaining to this part was over. Furthermore we decided to rewrite \( \text{GenList} \) to suit our need and ended up with

\[
\text{GenList a n = [a..n]} \implies \text{GenList n = [2..n]}
\]

This was done since 2 will be fix.

P2 concludes that the first element of the list formed in P1 is prime, therefore we must device a way to mark this as such. This situation calls for a function of type \( \text{num} \rightarrow \text{[num]} \rightarrow \text{[num]} \), where \( \text{num} \) is the type of the first element in the list, the first \( \text{[num]} \) corresponds to the type of the rest of the list after the first element was removed and the final \( \text{[num]} \) is the type of the original list. When WISER was queried with this the system responded with "No exact match found" as a result of which more general ones were then sought with the following six responses:

1. \( \text{drop} :: \text{num} \rightarrow \text{[a]} \rightarrow \text{[a]} \)
2. \( (\cdot) :: \text{a} \rightarrow \text{[a]} \rightarrow \text{[a]} \)
3. \( \text{insert} :: \text{a} \rightarrow \text{[a]} \rightarrow \text{[a]} \)
4. \( \text{eliminate} :: \text{a} \rightarrow \text{[a]} \rightarrow \text{[a]} \)
5. \( \text{postfix} :: \text{a} \rightarrow \text{[a]} \rightarrow \text{[a]} \)
6. const :: a → b → a

Examination of (:)'s source code showed that it was the required function. Now a way of distinguishing the first element from the rest of the list was possible (i.e. $2:[3,4,5,...,n] = [2,3,4,5,...,n]$).

P3 required the deletion of all elements from the list that are divisible by the first element. At this point we were wondering what would be the type of the function that could perform this task. One query type that came to mind was $\text{num} \rightarrow [\text{num}] \rightarrow [\text{num}]$, where $\text{num}$ is the type of the first element in the list, the first $[\text{num}]$ corresponds to the type of the rest of the list after the first element was removed and the final $[\text{num}]$ is the type of the resulting list free from multiples of the first element. Also $[\text{num}] \rightarrow [\text{num}]$ was considered as another possible query type, with the argument type corresponding to the original list (i.e. first element plus tail) and the result being the same as the previous query type. This was safe because from this query type we know how to obtain the first element form the list without having to physically separate them. The choice of which type to actually query with was simple since we know that the software repository can only obtain functions with more arguments than the queried type but not vice-versa, and that $\text{num} \rightarrow [\text{num}] \rightarrow [\text{num}]$ could produce the type $[\text{num}] \rightarrow [\text{num}]$ by freezing its first argument. Hence the query type was $[\text{num}] \rightarrow [\text{num}]$. The response to this query produced the function sieve :: $[\text{num}] \rightarrow [\text{num}]$ which was as a result of an exact match. The source code for sieve was as follows:

sieve (a:ax) = sieve [i|i<-ax; i mod a `~=` 0]

and its documentation showed us that it did exactly what we wanted our function to do. So from this it was not necessary to use the query type $\text{num} \rightarrow [\text{num}] \rightarrow [\text{num}]$. 

59
Table 11 Magic square with \( m = 5 \)

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>24</th>
<th>1</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>13</td>
<td>20</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>19</td>
<td>21</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>25</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

The final part requires us to repeat the process with the sequence resulting from P3, starting with P2. So what we did is take the first element off the list and append (using the function \( \cdot \) obtained earlier) it to the recursive call of sieve giving us a list of prime numbers. Also P4 gives us the base condition for this. Hence the solution the prime numbers problem will be

\[
\text{primes n} = \text{sieve \{2..n\}}
\]

\[
\text{sieve \{\} } = \{\}
\]

\[
\text{sieve \{a:ax\} } = \text{a : sieve \{i\mid i<?ax; i \mod a \neq 0\}}
\]

If you required all the prime numbers up the integer 150, then the answer is as follows:

Miranda primes 150

\[
\]

### 4.2.2 Magic square

The next application of WISER to aid in programming, focuses on solving the magic square problem. We will focus on solving this problem using Miranda functions by taking advantage of the software repository provided by WISER.

A magic square is an \( m \times m \) matrix of integers from 1 to \( m^2 \), where \( m \) is odd, such that the sum of every row, column and diagonal is the same. When \( m = 5 \) the resulting square is as shown in Table 11 with the common sum being 65. We will use Coxeter
### Table 12: A Program that solves the magic square problem

```haskell
magic m = error "Dimension Of Magic Table Must Be Odd", (m mod 2)=0
  = "[1]", m = 1
  = print([L|x<-[1..m];L<-(getLine x (sortList (makeList m))))], otherwise where
  getline x ax = [[i|(i,j)<- (sortTuple [(n,z)|(n,z)<- ax;y=x)]]]
  sortTuple [] = []
  sortTuple ((a,c):xs) = sortTuple [(k,m)|(k,m)<-xs; m<=c]
                              ++ [(a,c)]
                              ++ sortTuple [(k,m)|(k,m)<-xs; m>c]
  sortList [] = []
  sortList ((a,b,c):xs) = sortList [(k,l,m)|(k,l,m)<-xs; m<=c]
                             ++ [(a,b,c)]
                             ++ sortList [(k,l,m)|(k,l,m)<-xs; m>c]
  makeList m = theList (m, m^2-1)
  theList (a,b) = next [(b,(a div 2)+1,1)], if b=1
                   next (theList (a,b-1)), otherwise where
                   next [z] = (coor z):[z]
                   next (z:zs) = (coor z):zs
                   coor(n,x,y) = (n+1, x, y+1), if(n mod a) = 0
                                 = (n+1, x+1, a), if y = 1
                                 = (n+1, 1, y-1), if x = a
                                 = (n+1, x+1, y-1), otherwise
      print [] = []
      print (x:xs) = show x ++ "\n" ++ sh xs
```

rule for generating the magic square. This rule states that “start with 1 in the middle of
the top row; then go up and right assigning numbers in increasing order to empty squares;
if you fall off the square imagine the same square is tiling the plane and continue; if a
square is occupied, move down instead and continue.”

Table 12 shows the resulting Miranda program which reused a number of Miranda
functions both from exact match and functional generalization. Dealing with exact match, the following functions were easily obtain and identified as reusable candidates by examining their source code:

1. \( \text{mod} :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
2. \( (=) :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
3. \( \text{div} :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
4. \( (+) :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
5. \( (-) :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
6. \( (++) :: \text{num} \rightarrow \text{num} \rightarrow \text{num} \)
7. \( (:) :: a \rightarrow [a] \rightarrow [a] \)

We also needed functions to sort tuples based upon their elements, and as such queried the system with types \([(\text{num},\text{num})] \rightarrow [(\text{num},\text{num})] \) and \([(\text{num,\text{num,\text{num}}})] \rightarrow [(\text{num,\text{num,\text{num}}})]. \) However these failed in finding any exact match functions as a result of which more specific functions were then sought, but encountered the same fate as exact match. Next more general functions were requested with each producing the following response:

1. \( \text{tl} :: [a] \rightarrow [a] \)
2. \( \text{reverse} :: [a] \rightarrow [a] \)
3. \( \text{mkset} :: [a] \rightarrow [a] \)
4. \( \text{sort} :: [a] \rightarrow [a] \)
5. \( \text{init} :: [a] \rightarrow [a] \)
6. \( \text{elizezo} :: [a] \rightarrow [a] \)
7. \( \text{id} :: a \rightarrow a \)
8. \( \text{force} :: a \rightarrow a \)
Out of these, _sort_ was most prominent and upon examining its source code confirmed such. So with a little modification it was able to sort a list of tuples conforming to the desired way. Finally to print out the result a function that takes in a list of list of numbers and return a list of characters was needed. Upon querying with $[\text{num}] \rightarrow \text{[char]}$ yield no exact match functions but did however produce a more general function $\text{show} :: a \rightarrow \text{[char]}$ which we were able to incorporate into the _print_ function without doing any modifications to it.

This application reused a number of Miranda functions from the software repository during its construction, most of which needed no modifications, while others needed little. Obtaining these function was not difficult and furthermore they were integrated into the application with ease. The amount of mental effort required to take the application from the design phase to the source code phase with the aid of _WISER_ was fair. With $m = 15$ the application produced the following magic square:

```
Miranda magic 15

[122, 139, 156, 173, 190, 207, 224, 1, 18, 35, 52, 69, 86, 103, 120]
[138, 155, 172, 189, 206, 223, 15, 17, 34, 51, 68, 85, 102, 119, 121]
[154, 171, 188, 205, 222, 14, 16, 33, 50, 67, 84, 101, 118, 135, 137]
[170, 187, 204, 221, 13, 30, 32, 49, 66, 83, 100, 117, 134, 136, 153]
[202, 219, 11, 28, 45, 47, 64, 81, 98, 115, 132, 149, 151, 168, 185]
[218, 10, 27, 44, 46, 63, 80, 97, 114, 131, 148, 165, 167, 184, 201]
[9, 26, 43, 60, 62, 79, 96, 113, 130, 147, 164, 166, 183, 200, 217]
[25, 42, 59, 61, 78, 95, 112, 129, 146, 163, 180, 182, 199, 216, 8]
[41, 58, 75, 77, 94, 111, 128, 145, 162, 179, 181, 198, 215, 7, 24]
[57, 74, 76, 93, 110, 127, 144, 161, 178, 195, 197, 214, 6, 23, 40]
[73, 90, 92, 109, 126, 143, 160, 177, 194, 196, 213, 5, 22, 39, 56]
[89, 91, 108, 125, 142, 159, 176, 193, 210, 212, 4, 21, 38, 55, 72]
[105, 107, 124, 141, 158, 175, 192, 209, 211, 3, 20, 37, 54, 71, 88]
[106, 123, 140, 157, 174, 191, 208, 225, 2, 19, 36, 53, 70, 87, 104]
```

4.2.3 Discussion

We were able to solve the prime numbers problem by reusing existing software components through simplistic modifications. It basically uses two functions, but required us to perform three retrieval process to confirm such. To locate $\text{num} \rightarrow \text{num} \rightarrow [\text{num}]$
required the examination of 11 out of the 56 two argument records. Also the type \texttt{num} \rightarrow \texttt{[num]} \rightarrow \texttt{[num]} did not exist at the time of querying and required the examination of 11 records to confirm such and was subsequently inserted.

In the examples on retrieval given in section 4.1.1 the number of records examined before finding an exact match for type \texttt{[a]} \rightarrow \texttt{a} was 4 as compared to the 156 records that exist in the software repository. Also for type match on \texttt{[num]} \rightarrow \texttt{[num]} required the examination of only 6 records out of the 156. In the application on magic square a retrieval query type of \texttt{[[num]]} \rightarrow \texttt{[char]} was applied to obtain one of the functions used, however this failed in yielding an exact match since no records existed with that type. It took the examination of only 14 records before the system could determine such (for this case the search went all the way down to the leaf nodes). With the same application and query types \texttt{[(num,num)]} \rightarrow \texttt{[(num,num)]} and \texttt{[(num,num,num)]} \rightarrow \texttt{[(num,num,num)]} it required the examination of 7 and 8 records respectively, before figuring out that no type match existed. The reason for these not being identical is that once we tried to retrieve \texttt{[(num,num)]} \rightarrow \texttt{[(num,num)]} and found that it does not exist, it was inserted, as a result of which the retrieval of \texttt{[(num,num,num)]} \rightarrow \texttt{[(num,num,num)]} yielded the examination of one more record. For a final and unrelated (to these applications) statistic, it took the examination of 18 out of 148 records before finding the type \texttt{(num} \rightarrow \texttt{bool)} \rightarrow \texttt{[num]} \rightarrow \texttt{[num]} and after the inserting of 8 more records into the structure, took the examination of only 6 records to find it once again. It is worth noting here that some and not all of the 8 records inserted were responsible for this decrease in the search space.

In all of the cases above in which we did not find an exact type match, if we were to once again query \textit{immediately} with those same types an exact type match would be found by examining one record more than we did previously. The word \textit{immediately} was used to acknowledge the fact that the software base is growing and as such the insertion
of other types could decrease or increase the number of records examined.
Chapter 5 Conclusions and Future Work

This chapter explains the conclusions formed as a result of this thesis and outlines some important extensions.

5.2 Conclusions

In this thesis we have investigated a structured software base to support reuse-oriented program development, and developed a prototype reusable software base system, WISER. This system rallied around the concept of maintaining an evolving structured software base with various tools to allow for effective insertion, deletion and retrieval of software components. It further described how components represented by types were interconnected by polymorphic and extra-argument relations, thereby allowing easy access to general, specific and extra-argument functions. A way of storing these types along with their corresponding function(s) and access paths was also looked at along with a way of browsing the structure. We also focused on the insertion of types into the software base when the retrieval process failed in finding an exact type match. Furthermore this thesis was centered around curried functions. If the user wants to retrieve a function that sums two list of numbers together, then his/her query type should be $\text{[num]} \rightarrow \text{[num]} \rightarrow \text{[num]}$ and not the uncurried version $(\text{[num]}, \text{[num]}) \rightarrow \text{[num]}$. This uncurried type would retrieve components, but not the required one, since its argument is treated as a tuple. It is also difficult for users to know the order of arguments associated with functions in the software repository, and as such WISER was designed to take into consideration the ignorance of the user to argument order. Finally, a variation of the breadth-first search algorithm was used in the acquisition of more general, more specific and extra-argument
functions, so as to maintain some natural ordering. All user interaction with the system are window-based so as to allow for easy input/output operations for the users.

In chapter 4 we explained how many records needed to be examined in order to find various records corresponding to our query type. We furthermore showed that not all records needs to be examined to determine whether the record exists or not. The number of records examined for the deletion process is the same as that for retrieval. The insertion of a component could also be accomplished in the same amount of examination as the retrieval, for the case of the type existing already. However if the type does not exists, then in the worst case the examination could span the entire software library, since we are also going to insert types that are formed by various combinations of arguments from the original type. But nevertheless this could decrease as the number of records in the software base increases, because the chances of some of its spawned types existing increases. The complexity of the retrieval process for WISER proves to be linear, in terms of the number of types, for the worst case but on the average is better and can be best described as having an amortized complexity.

Initially the graph was shallow but as more records were inserted it became more proportioned. The maximum number of parent or children for a given record in the software base is 12 and could increase or decrease depending upon the types of other records inserted.

Based upon the work carried out in this thesis a number of conclusions were formed as listed below:

1. Using WISER with its structured software repository provided a very useful tool for reuse-oriented program development, especially reuse-in-the-large.
2. The structured software base approach supports automatic retrieval.
Conclusions and Future Work

3. More useful functions could be retrieved in the form of relax matches.

4. WISER could be used as an aid to program in a functional programming course.

5. Automatic software retrieval tools like WISER would be essential in facilitating software reuse for huge repositories of components.

6. The structured software base approach can be extended to non-functional software components like logical and object-oriented ones.

7. In average-case, performance will be better than linear search.

8. WISER allows the user to browse the structure of reusable components with the aid of a graphical interface.

9. WISER maintains an evolving software repository.

In concluding the user can ultimately access all of the components with $n$ arguments by querying with the root type for $n$ arguments and finding more specific components.

5.3 Future Work

Retrieval of software components from a software library using types as search keys allows for the retrieval of suitable candidate functions. However the number of components retrieved is unpredictable ranging from a few to many. With the structured software base approach the functions retrieved were broken up into four categories. But these categories also have an unpredictable number of components. Therefore this approach needs to be complemented with other techniques so as to further narrow down the retrieved functions to the most appropriate one(s).

In keeping with the theme of using semantic properties of functions, components in the software base can be stored with some properties in the form of informal partial specification attached to them. Then the retrieval of components can be accomplished by using both the types and these properties, some of which are in the form of the strictness
of the function, lifetime of arguments, the one-to-one or onto nature of the function and information about lists. Sampling behavior can then be applied to the functions retrieved from above.

Another area of future work lies with retrieval based on functional composition. That is, we can construct our required function by using a number of subsidiary functions. Let's say we are looking for a function that counts the number of distinct elements in a list of numbers, and failed in retrieving any useful one(s) based on the conventional retrieval methods mentioned. Then we should pursue the idea of functional composition, as in composing \texttt{removeDuplicates :: [num] \to [num]} with \texttt{count :: [num] \to num} (i.e. \texttt{count \after removeDuplicates}). This composition is based on types, so that composing functions with types \texttt{a \to c} and \texttt{c \to b} should give the resulting function with type \texttt{a \to b} and can be extended to the composition of \textit{n} functions.
Appendix A: Sample Of Software Library

A.1 One-Argument file

(a->b) failed \{ 250, 500, 750, 1250, 2500, 4250, 6250, 6750, 9250, 9500, 11000, 11250, 11750, 16000 \} \{ 7500, 12750 \}

(b) \{ a \} \{ 0 \} \{ 1500, 4500, 6500, 9000, 11500 \} \{ 250, 500, 3500, 13250 \}

(a->a) \{ id, force \} \{ 0 \} \{ 1000, 4250, 12000, 13750 \} \{ 250, 750, 3750, 4000, 6750, 7000 \}

(a->a) \{ tl, reverse, mkset, sort, init, elizer, boolSum \} \{ 500, 1500, 3500, 8500, 12750, 14250 \} \{ 1000, 5500, 6000, 8000 \}

(a->bool) \{ a \} \{ 16000 \} \{ 14500 \} \{ 1000 \}

(b) \{ a \} \{ 250, 16000 \} \{ 1000, 2500, 2750, 3000, 7750, 8250, 8750, 10500, 21000, 13500, 15000, 15500 \} \{ 1250, 1500, 2000, 2500, 4500 \}

(a->a) \{ repeat \} \{ 16000 \} \{ 7750, 12500 \} \{ 8000, 17500, 5250, 7750 \}

(a->b->a) \{ a \} \{ 2250 \} \{ 1500, 1750 \}

(a->b) \{ a, b, c \} \{ 1500 \} \{ 2750, 3000 \}

(b) \{ a, b, c \} \{ 1500 \} \{ 2750, 3250 \}

(a) \{ a, b, c \} \{ 1500 \} \{ 3000, 3250 \}

[char] \{ read, filemode, getenv \} \{ 1000, 13250 \} \{ 4250 \}

(num->char) \{ shownum, spaces \} \{ 5750, 13250 \} \{ 4250, 4750 \}

(a->b) \{ b \} \{ 16000 \} \{ 5500 \} \{ 4500 \}

(num->num) \{ neg, sin, sqrt, abs, cos, entier, exp, log, log10, arctan, fib, fact, listdiv \} \{ 500 \} \{ 8250 \}

(a->a) \{ hd, last, limit, max, min, middle \}

[250] \{ 5250, 8250, 10000 \} \{ 5750, 8500 \}

[[char] \{ sysm \} \{ Appendfile, Closefile, Stderr, Stdout, System \}

[9000] \{ 5000 \}

(num->sysm) \{ Exit \} \{ 11750 \} \{ \} \}

([bool] \{ bool \} \{ and, or \} \{ 4500, 6500 \} \{ \} \}

(a->a) \{ a \} \{ 4000 \} \{ 14750 \} \{ 5250 \}

(num->a) \{ 11750, 16000 \} \{ 12500, 13000 \} \{ 5500, 7750 \}

(a->a->a->a) \{ 750, 3250 \} \{ 5750 \}

(a->bool) \{ 0 \} \{ 6500, 7000, 7250, 12000, 12250, 15750 \} \{ 6250, 9000, 12500 \}

(a->bool) \{ 250, 6250 \} \{ 5250, 14000 \} \{ 6250, 10750, 12250 \}

(char->num) \{ code \} \{ 0 \} \{ \} \}

(char->bool) \{ digit, letter \} \{ 6250 \} \{ \} \}

(num->bool) \{ integer, odd, even, prime \} \{ 6250 \} \{ \} \{ 11500 \}

([char] \{ [char] \} \{ lay, layn \} \{ 8250, 13250 \} \{ \} \}

([char] \{ [char] \}) \{ lines \} \{ 1500, 1750 \} \{ \} \}

([char] \{ num \} \{ numval \} \{ 9000, 11500 \} \{ \} \}

([a] \{ a \}) \{ listConata \} \{ 1500, 4500 \} \{ 7500 \} \{ \}

([a] \{ a \}) \{ transpose \} \{ 1000 \} \{ \} \}

([a] \{ num \}) \{ index \} \{ 1500 \} \{ 12750 \} \{ \}

([char] \{ a \}) \{ error \} \{ 250 \} \{ 4750, 8000, 14000 \} \}

(a->a) \{ fst \} \{ 0 \} \{ \} \}

(a->a) \{ snd \} \{ 0 \} \{ \} \}

70
A.2 Two-Argument file

```
(a->b->c) { } {7250, 7500, 9000, 12500, 12750, 13000} {0}
(a->b->a) { } {7500, 12750} {6000} {250, 1250}
((a->b->a) -> [b->a]) {12750} {5750} {250}
((a->b->a) -> [a->a]) {7500} {250}
((a->bool) -> [a->a]) filter, takewhile, dropwhile {7500, 13500} {11000} { }
(a->b->a) { } {13500} {8000} {500}
((a->b->a) -> [b->a]) {2250} {500}
((a->b->a) -> [a->a]) {13000} {500}
(a->b->c) { } {13250, 13500} {2750, 3000, 3250, 6000, 6500} {750, 2000}
((a->b->c) -> [b->c]) {13500} {1500} {750}
((a->b->c) -> [a->c]) {13500} {750}
(b->c) -> [a->(a,b,c)] {2000} {1000}
(a->c) -> [a->(a,b,c)] {2000} {1000}
(a->b->b) -> [a->b] {12750} {5750} {1250}
((a->b->b) -> [b->b]) {750} {1250}
(a->a->a) [min2, max2] {7500} {6000, 8250, 8750, 10000} { }
(num->[char] -> [char]) {cjustify, rjustify, ljustify} {5500}
```
A.3 Three-Argument file
Appendix B: Program Listing

/* .................................................................
This program is constructed so as to allow the user to retrieve, insert and
delete miranda function(s) of a given type.
................................................................. */

#include <Xm/Xm.h>
#include <Xm/List.h>
#include <Xm/Scale.h>
#include <Xm/Separator.h>
#include <Xm/BulletinB.h>
#include <Xm/Text.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <Xm/Form.h>
#include <Xm/CascadeB.h>
#include <Xm/SelectionB.h>
#include "wiserbmt"
#include <Xm/DrawingA.h>

#define APPLY 1
#define QUIT 2
#define RESET 3
#define HELP 4
#define GENERAL 5
#define YES 1
#define NO 2
#define SPECIFIC 6
#define FREEZE 7
#define EXACT 8
#define SYMBOLS 1
#define ID 2
#define SPACE 3
#define FUN_OP 4
#define RECORD 3
#define NAME 1
#define OK 1
#define CANCEL 2
XtAppContext context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel, form, label, menu_bar, sep, dialog, dialog1;
Widget retrieve_menu, file_menu;
Widget retrieve_item, save_item, empty_item;
Widget quit_item;
Widget browse_menu, insert_menu;
Widget browse_item;
Widget version_item;
Widget version_menu;
Widget delete_menu, insert_item, delete_item;
Widget help_menu, about_item, help_item;
Widget list, list1, list2, list3, list4, list5, list6, list7, list8, list9, list10.
Program Listing

```c
char *base_type[] = {"char","bool","int","real","num","sysm"};
int flag, num,exact_match, value,check_flag, Exit_flag;
long int SSoffset[50], SGoffset[50], SSoffset[50], Ft, backOffset;
int SIndex, SGindex, ok, c, val;
char *track[100];
char *storage[1000];
char result[50]; /* the user must enter the brackets ( & ) */
char function[](50), typo_browser[1000];
int general_index, exact_index, specific_index, freeze_index;
char *general_functions[2][100], *exact_functions[2][100];
char *specific_functions[2][100], *freeze_functions[2][100];

char *filename[9] = {"a0","a1","a2","a3","a4","a5","a6","a7","a8"};

typedef char std[100];
stn gen_browser[5][20], spec_browser[5][20], freeze_browser[5][20];
int gen_bro[5][20], spec_bro[2][20], freeze_bro[2][20];

int gen_num, spec_num, bro, free_num_bro;

void find_freeze(char t4[100], FILE *fp, int arg_num);
void freezone(char t4[100], int arg_num);
void search_type(int count, FILE *fp, char *list[10], long int spec[50],
int arg_num);
void retrieve(char *Slist[10], int arg_num);

char *argument[] = {"Argument1","Argument2","Argument3","Argument4",
"Argument5","Argument6","Argument7","Argument8"};

int browse_arg = 0;
static XColor Red, Black, Green, Blue, White, Navy, Orange, Yellow, Pink, Magenta, Cyan,
Brown, Grey, LimeGreen, Turquoise, Violet, Wheat, Purple;

/* ..............................SYSTEM SOURCE CODE WRITTEN IN C................
.................................-------------------------------------------------------------------- */
/* This function is used to remove unwanted storage */
void free_storage()
{
  int i;
  for(i=0;value;i++)
    free(storage[i]);
  return;
} /* ................................. */
/* This function is used to remove unwanted storage */
void free_function_storage()
{
int i;
for(i=0;i<general_index;i--)
{
    free(general_functions[0][i]);
    free(general_functions[1][i]);
}
for(i=0;i<specific_index;i--)
{
    free(specific_functions[0][i]);
    free(specific_functions[1][i]);
}
for(i=0;i<exact_index;i--)
{
    free(exact_functions[0][i]);
    free(exact_functions[1][i]);
}
for(i=0;i<freeze_index;i--)
{
    free(freeze_functions[0][i]);
    free(freeze_functions[1][i]);
}
return;

/* ................................................................. */
/* This function is used to remove unwanted storage */
void remove_track()
{
    int j;

    for (j=0;j<num;j++)
    {
        free(track[0][j]);
        free(track[1][j]);
    }
}

/* ................................................................. */
/* Prevents us from outputting duplicate functions. */
int multi_memb(char *string, char *list[2][100], int length)
{
    int i;
    for (i =0; i<length;i++)
    if (strcmp(string,list[1][i]) == 0)
        return 1;
    return 0;
}

/* ................................................................. */
/* The member function determines whether or not a string exists in a list of strings */
int member(char *string, char *list[4], int length)
{
    int i;
    for (i =0; i<length;i++)
    if (strcmp(string,list[i]) == 0)
        return 1;
    return 0;
}
Program Listing

/* The Token function takes in as arguments a character pointer to an
expression along with a character pointer to a list of character
pointers. It takes the expression and tokenize it according to known
tokens, eg. int, bool, ->, [...]. The tokens are then stored so that a
list of pointers indicates each storage location. It returns the total
number of tokens. */

int Token(char *expr, char **tokarray)
{
    char temp[5];
    char *ptr = temp;
    int control;
    int val = 0;
    char p[3];

    while (*expr != '\0')
    {
        temp[0] = '\0';
        ptr = temp;

        if (isalpha(*expr))
            control = 2;
        else
            if (strchr("[]", *expr) != NULL)
                control = 1;
            else
                if (strchr("", *expr) != NULL)
                    control = 4;
                else
                    if (*expr == ' ' || *expr == 't')
                        control = 3;
                    else
                        control = 5;

        switch (control) {
        case SYMBOLS :
            p[0] = *expr; p[1] = '\0';
            tokarray[val] = (char *)malloc(strlen(p)*sizeof(char) + 1);
            strcpy(tokarray[val], p);
            val++;
            break;
        case ID :
            while ((*expr != '\0') && (isalpha(*expr)))
            {
                *ptr = *expr; expr++;
                ptr++;
            }
            expr++;
            *ptr = '\0';
            if ((strlen(temp) == 1) || ((strlen(temp) > 2) &&
                (member(temp, base_type, 6))))
            {
                tokarray[val] = (char *)malloc(5*sizeof(char));
                strcpy(tokarray[val], temp);
            }
            else
            {
                ...
printf("ERROR : Syntax error in input\n");
exit(1);
}
val--;
brack;
case SPACE :
brack;
case FUN_OP :
p[0] = *expr;
expr++;  
if (strchr("->", *expr) == NULL)
{
    printf("ERROR : Syntax error in input\n");
    exit(1);
}
tokarray[val] = (char *)malloc(strlen(p)*sizeof(char) + 1);
strcpy(tokarray[val], p);
val++;
brack;
default :
    printf("ERROR : Syntax error in input\n");
    exit(1);
expr++;  
}
return val;

/*  
*  This function takes in a list of tokens and returns the first argument  
*  along with the number of tokens that makes it up. Also, the remainder  
*  of the token list and its number of arguments is returned.  
*/
int arg_rest(char ***argument, char ***list, int *num, char **oper)
{
    int counter;
    int end_pos = 0;
    counter = 0;

do {
    if ((strcmp(*list)[end_pos], "(" ) == 0)
        || (strcmp(*list)[end_pos], "[" ) == 0)
        counter++;
    else
        if ((strcmp(*list)[end_pos], ")" ) == 0)
            || (strcmp(*list)[end_pos], "]" ) == 0)
                counter++;
            end_pos++;
}while ((end_pos != *num) && ((strcmp(*list)[end_pos], "," ) != 0)
        || (counter != 0) && ((strcmp(*list)[end_pos], ".->" ) != 0)
        || (counter != 0));

    counter = 0;
    if ((end_pos != *num) && ((strcmp(*list)[end_pos], ".->" ) == 0)
        || (strcmp(*list)[end_pos], "," ) == 0))
    {
        strcpy(*oper, (*list)[end_pos]);
/* This function takes in two list of tokens and the number of tokens in each. It then performs unification on both list, argument by argument recursively. It uses a global array called track[1][1], to keep track of the substitutions. And, a global flag to determine a subset of not unifiable. */

void unify(char **tempstak1, int v1, char **tempstak2, int v2)
{
    char **t1stak, **t2stak;
    int pos1, i, pos2;
    char operator1[3] = "=";
    char operator2[3] = "=";
    char *op2 = operator2;
    char *op1 = operator1;

    if ((v1 == 0) && (v2 == 0))
        printf("\n");
    else if ((v1 == 1) && (strlen(tempstak1[0]) == 1) && isalpha(*tempstak1[0]))
    {
        if (v2 != 0)
        {
            l = 0;
            track[0][num] = (char *)malloc(strlen(tempstak1[0]) + sizeof(char) - 1);
            strcpy(track[0][num], tempstak1[0]);
            track[1][num] = (char *)malloc(50 + sizeof(char));
            strcpy(track[1][num], "=");
            while (i < v2)
            {
                strcat(track[1][num], tempstak2[i++]);
            }
            num++;
        }
    else
        flag = 0;
    else
        if (member(tempstak1[0], base_type, 6) && (strcmp(tempstak1[0], tempstak2[0]) == 0))
            printf("\n");
        else
            if (((strcmp(tempstak1[0], "]") == 0) && (strcmp(tempstak2[0], "]") == 0))
            || ((strcmp(tempstak1[0], ")") == 0) && (!strcmp(tempstak2[0], ")") == 0))
                v1 = 2;

78
v2 := 2;
 *tempstak1++;
 *tempstak2++;

 pos1 = arg_rest(&t1stak, &tempstak1, &v1, &op1);
 pos2 = arg_rest(&t2stak, &tempstak2, &v2, &op2);

 if (v1 != 0)
   {
     v1 -= 2;
     *tempstak1--;
     strcpy(tempstak1[0], "(');
     strcpy(tempstak1[v1-1], ")");
   }
 if (v2 != 0)
   {
     v2 -= 2;
     *tempstak2--;
     strcpy(tempstak2[0], "(');
     strcpy(tempstak2[v2-1], ")");
   }
 if (strcmp(op1, op2) == 0)
   {
     unify(t1stak, pos1, t2stak, pos2);
     unify(tempstak1, v1, tempstak2, v2);
   }
 else
   {
     flag = 0;
   }
 else
   {
     flag = 0;
   }
 return;
}

/* ------------------------------------------------------ */
/* Tests whether the types are unifiable and if so, are they exact. flag 
and exact_match are both global variables. */
void test()
{
  int i, j;
  exact_match = 1;

  if (num == 1)
    if (strlen(track[1][0]) != 1)
      exact_match = 0;
  if (flag != 0)
    {
      for (i=0; i < (num - 1); i++)
        for (j=i+1; j < num; j++)
          {
            if (strcmp(track[0][i], track[0][j]) == 0)
              {
                if (strcmp(track[1][i], track[1][j]) != 0)
                  {
                    flag = 0;
                    exact_match = 0;
                    i = num;
                    j = num - 1;
                  }
              }
        }
    }
else
    if (strlen(track[1][i]) != 1)
        exact_match = 0;
    else
        if (((strcmp(track[1][i], track[1][j]) == 0) ||
            (strlen(track[1][i]) != 1) || (strlen(track[1][j]) != 1))
            exact_match = 0;
    }

return;

/* ................................................................. */

void interchange(char **argum, int k, int i)
{
    char temp[20];
    strcpy(temp, argum[k]);
    strcpy(argum[k], argum[i]);
    strcpy(argum[i], temp);

    return;
}

/* ................................................................. */

/* Finds all possible combinations of argument order. This allows us to
find instances regardless of argument order. The different combinations
of argument order are stored in the global variable called "storage". */

void arg_comb(char *A[10], int k, int n)
{
    int i, j;
    char *B[10];

    if (k == n)
    {
        storage[value] = (char *)malloc(100*sizeof(char));
        strcpy(storage[value], "(");
        j = 0;
        while(j < (n+1))
        {
            strcat(storage[value], A[j++]);
            strcat(storage[value], "," +");
            strcat(storage[value], result);
            strcat(storage[value], ")" +");
            value++;
            return;
        }
    }

    for (j=0; j < (n+1); j++)
    {
        B[j] = (char *)malloc(30*sizeof(char));
        strcpy(B[j], A[j]);
    }

    for (i=k; i < (n+1); i++)
    {

interchange(A,k,1);  
arg_comb(A,k+1,n);  
for (j=0; j < (n-1); j++)  
    strcpy(A[j],B[j]);  
}

/* This function takes in two types, "s1" & "list", along with the number  
of arguments, which must be same in both. It then calls the required  
functions so as to determine whether one is an instance of the other,  
based on the variable "order". If order = 0 then it determines whether  
"list" is an instance of "s1" and vice versa for order = 1. */  
void instance (char s1[100], char *list[10], int arguments, int order)  
{
    int valt1, valt2;  
    char a1[100];  
    char a2[100];  /* user Query*/  
    char *a2P = a2;  
    char *a1P = a1;  
    char *tlist[30], *t2list[30];  
    char **t1 = tlist;  
    char **t2 = t2list;  
    int j = 0;

    if (arguments == 0)  
    {  
        if (check_flag == 0)  
        {  
            value = 1;  
            storage[0] = (char *)malloc(100*sizeof(char));  
            strcat(storage[0], result); check_flag =1;  
        }  
        flag = 1;  
        num = 0;  
        if (order == 0)  
        {  
            strcpy(a2,storage[0]);  
            strcpy(a1,s1);  
        }  
        else  
        {  
            strcpy(a1,storage[0]);  
            strcpy(a2,s1);  
        }  
        valt1 = Token(a1P,tlist);  
        valt2 = Token(a2P,t2list);  
        unify(t1,valt1,t2,valt2);  
        test();  
        remove_track();  
        return;  
    }

    if (check_flag == 0)  
    {  
        value = 0;arguments...;  
        arg_comb(list,0,arguments);  
        check_flag = 1;  
    }
for (j=0;j<value;j++)
{
  flag = 1;
  num = 0;
  if (order == 0)
  {
    strcpy(a2,storage[j]);
    strcpy(a1,s1);
  }
  else
  {
    strcpy(a1,storage[j]);
    strcpy(a2,s1);
  }
  a2Pt = a2;
  a1Pt = a1;
  valt1 = Token(a1Pt,t1list);
  valt2 = Token(a2Pt,t2list);
  t1 = t1list;
  t2 = t2list;
  unify(t1,valt1,t2,valt2);
  test();
  remove_track();
  if (flag == 1)
    return;
}
return;

/* .............................................................. */
/* Given a string of long int numbers separated by a comma, this function */
/* extracts these long int numbers and stores it in an array of long int. */
/* It also, returns the number of numbers found. */
void gen_offset(char temp2[200],long int *gen,int *num)
{
  int j = 0;
  long int offset;
  char temp[200],names[12],b[200];
  strcpy(b,temp);

  if (strcmp(temp2," ") == 0)
  {
    *num = j;
    return;
  }

  while (strcmp(temp2, "DONE") != 0)
  {
    if (sscanf(temp2,"%[^,], %s ",names,temp) == 2)
    {
      offset = atol(names);
      gen[j] = offset;
      strcpy(temp2,temp);
    }
    else
    {
      sscanf(temp2,"%[^,]*",names);
      offset = atol(names);
    }
  }
gen[j] = offset;
strcpy(temp2, "DONE");

/*
* Given a string of names separated by a comma, this function extracts
* these names and stores it in an array of strings. It also, returns the
* number of names found. */
void function_names(char templ[200], char **nam, int *num)
{
    int i = 0;
    char templ[200], names[20], b[200];
    strcpy(b, templ);

    if (strcmp(templ, "") == 0)
        {*num = i;
         return;
        }

    while (strcmp(templ, "DONE") != 0)
    {
        if (sscanf(templ, "%[", names, templ) == 2)
        {
            nam[i] = (char *)malloc(strlen(names) * sizeof(char) + 1);
            strcpy(nam[i], names);
            strcpy(templ, temp);
        }
        else
        {
            sscanf(templ, "%", names);
            nam[i] = (char *)malloc(strlen(names) * sizeof(char) + 1);
            strcpy(nam[i], names);
            strcpy(templ, "DONE");
        }
        i++;
    }
    *num = i;
    strcpy(templ, b);
}

/*
* Inserts the root of each file based on the number of arguments. */
void insert_root(char name[15], char line[250])
{
    FILE *fp;
    int length, i;

    length = 250 - strlen(line);

    for (i = 0; i < length; i++)
        strcat(line, " ");

    if ((fp = fopen(name, "r")) == NULL)
exit(1);

fputs(line, fp);
fclose(fp);
}

/* This function aids the insertion of a type by returns the offset to
   which the (n-1)-argument function links to the n-argument function. F needs
   corresponding to the offset of the n-arg. function just inserted.
Note: n > 1.
If F = 0 then
   the freezing component (FC) of (n-1)-arg. type is 1;
else
   if FC = [ ] then
      insert F in FC
   else
      for i = fc_k
      if fc_i-type is an instance of F-type then
         do nothing return
      else
         if F-type is an instance of fc_i-type then
            remove fc_i and insert F.
   end for
if exhausted FC and no update to it then append F to it. */
void freeze_extra_arg(int arg_num, char line[250], char t4[200], char *Flist[10])
{
    long int freeze[50];
    long int *freezePtr = freeze;
    FILE *fp;
    char temp[12], templine[250];
    int i, f, j;
    char Etype[100];

    gen_offset(t4, freezePtr, &f);

    if (F = 0)
    {
        strcat(line, t4);
        return;
    }

    arg_num++;
    if ((fp = fopen(filename[arg_num], "r+b")) == NULL)
        exit(1);
    for(i = 0; i < f; i++)
    {
        fseek(fp, freeze[i], 0);
        fgets(templine, 250, fp);

        scanf(templine, "%[^,],%", Etype);

        check_flag = 0;
        for(j = 0; j < value; j++)
            free(storage[j]);

        }
Program Listing

instance(Etype,Flist,arg_num,1):

if ((F==freeze[i]) || (flag == 1))
{
   strcat(line,t4);
   return;
}
else
{
   instance(Etype,Flist,arg_num,0);
   if (flag == 1)
   {
      freeze[i] = F;
      for(j=0; j<F; j++)
      {
         sprintf(temp, "%d", freeze[i]);
         strcat(line,temp);
         if (j < F-1)
            strcat(line,".");
      }
   }
}
for(j=0; j<F; j++)
{
   sprintf(temp, "%d", freeze[i]);
   strcat(line,temp);
   strcat(line,".");
}
sprintf(temp, "%d", F);
strcat(line,temp);
return;

/* This function aids in the insertion by updating the relevant components of various types to reflect whether or not they are more general or more specific than the one being inserted. It determines whether an exact type already exists and if it does then the function name is inserted if it does not exists already. After calling insert all general types are linked directly and/or indirectly to the inserted type. */
void insert(char *Slist[10], long int St, long int oldT, long int Et,
   FILE *fp, char name[20],int argument_num, char *Flist[10])
{
   long int general[50],spec[50];
   long int *genPtr = general;
   long int *specPtr = spec;
   char *nam[50];
   char **namPtr = nam;
   char Etype[100], t1[200], t2[200], t3[200], t4[200], temp[200];
   int length, i,j,k,l,b;
   int exhausted,flag1,exact;
   char line[250];
   long int offset;

   if (Exit_flag == 1)
      return;

   offset = 0;
   offset
   offset
fseek(fp, Et, 0);

if ((Et = ftell(fp)) != 0L)
  fgets(line, 250, fp);

if (sscanf(line, "%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]%[^\n]",
    Etype, t1, t2, t3, t4) != 5)
  { printf("end of file read pass
\n");
    return;
  }

gen_offset(t2, genPtr,&j);
gen_offset(t3, specPtr,&k);

exhausted = k;

instance(Etype, Slist, argument_num, 0);

flag1 = flag; exact = exact_match;

if (flag == 1)
  {
    if (exact_match == 1)
    {
      function_names(t1, namPtr,&i);
      strcpy(line, Etype); strcat(line, ":");
      if (member(name, nam.i) == 1)
      {
        strcat(line, t1);
        XtManageChild(ins_mat);
      }
    else
      if (strcmp(name,"\n") == 0)
        strcat(line, t1);
      else
        {
          if (i == 0)
            strcat(line, name);
          else
            {
              strcat(line, t1);
              strcat(line, ":\n");
              strcat(line, name);
            }
        }
    
    strcat(line, ":\n");
    strcat(line, t2); strcat(line, ":\n");
    strcat(line, t3);
    strcat(line, ":\n");
    freeze_extra_arg(argument_num, line, t4, Flist); check_flag = 0;
    for (b=0; b<value; b++)
      free(storage[b]);
    strcat(line, ":\n");
    length = 250 - strlen(line);
    for (i = 0; i < length; i++)
      strcat(line, " ");
  }
fclose(fp,Et,0):
  if ((Et = ftell(fp)) != -1L)
    fputc(line.fp):
  Exit_flag = 1:
  length = strlen(line):
  return;
else
  for(length = 0; length < k:length++)
  {
    oldT = Et:exhausted:-;
    Et = spec[length];
    insert(Slist, St, oldT, Et, fp, name, argument_num, Flist):
    Et = oldT:
  }
else
  {
    instance(Etype, Slist, argument_num, l):
    if (flag == 1)
    {
      flag = 0;
      for (i=0;i<SIndex;i++)
        if (SSoffset[i] == Et)
          {
            flag = 1;
            i = SIndex:
          }
      if (flag == 0)
      {
        SSoffset[SSIndex] = Et:
        SSIndex++:
      }
      flag = 0;
      for (i=0;i<SGIndex;i++)
        if (SGoffset[i] == oldT)
          {
            flag = 1;
            i = SGIndex:
          }
      if (flag == 0)
      {
        SGoffset[SGIndex] = oldT:
        SGIndex++:
      }
    backOffset = Et:
    strcpy(line,Etype):
    strcat(line,":“); strcat(line,t1); strcat(line,"*");

    b= 0;
    for(i=0; i<j;i++)
      if (general[i] == oldT)
        general[b++] = general[i]:
    flag = 0;
    for (i=0;i<b;i++)
      if (general[i] == St)
  {
flag = 1;
i = b;

} else if (flag == 0) {
general[b++] = St;
for (i = 0; i < b; i++) {
    printf(temp, \"%d\", general[i]);
    strcat(line, temp);
    if (i < b - 1)
        strcat(line, ",\"\");
}

strcat(line, \\"\"; strcat(line, t3));
strcat(line, \\"\"; strcat(line, t4); strcat(line, \\"\"));
length = 250 - strlen(line);
for (i = 0; i < length; i++)
    strcat(line, \\",\"\");
fseek(fp, Et, 0);
if ((Et = ftell(fp)) != -1L)
    fputs(line, fp);

fseek(fp, oldT, 0);
if ((oldT = ftell(fp)) != -1L)
    fgets(line, 250, fp);

if (sscanf(line, \\"%\[\]%\[\]%\]%%%\[\]%\]%%%\[\]%\]%%%\[\]%\]%%%\[\]%\"
    , &type.t1, t2, t3, t4) != 5) {
    printf("ERROR: Corrupted file\n");
    gen_offset(t3, specPtr, &l);
    strcpy(line, &type);
    strcat(line, \\"\"; strcat(line, t1); strcat(line, \\"\"));
    strcat(line, t2); strcat(line, \\"\"));
    b = 0;
    for (i = 0; i < l; i++)
        if (spec[i] != Et)
            spec[b++] = spec[i];

flag = 0;
for (i = 0; i < b; i++)
    if (spec[i] == St)
        flag = 1;
i = b;

} else if (flag == 0)
    spec[b++] = St;
for (i = 0; i < b; i++) {
    printf(temp, \"%d\", spec[i]);
    strcat(line, temp);
    if (i < b - 1)
        strcat(line, ",\"\");
}

strcat(line, \\"\"; strcat(line, t4); strcat(line, \\"\"));
length = 250 - strlen(line);
for (i = 0; i < length; i++)

```c
strcat(line, " ");
fsseek(fp.oldT, 0);
if ((oldT = ftell(fp)) != -1L)
    fputs(line.fp);
ok = 0;
}
}

if (Exit_flag == 1)
    return;

if ((flag1 == 1) && (exact != 1) && (exhausted == 0) && (ok == 1))
{
    flag = 0;
    for (i = 0; i < k; i++)
    {
        fsseek(fp.spec[i], 0);
        fgets(line, 250, fp);
        sscanf(line, "%["\[]", temp);
        instance(temp, Slist.argument_num, 0);
        if (flag == 1)
            i = k;
        if (flag == 0)
            for (i = 0; i < SGindex; i++)
                if (SGoffset[i] == Et)
                    flag = 1;
        i = SGindex;
    }
    if (flag == 0)
    {
        SGoffset[SGindex] = Et;
        SGindex++;
    }
    backoffset = Et;
    sprintf(temp, "%d", Et);
    strcpy(line.Etype); strcat(line,"[":strcat(line,t1);
    strcat(line,"("): strcat(line,t2); strcat(line,"\"]");
    if (strcmp(t3, ")") != 0)
    {
        if (strstr(t3, temp) == NULL)
            { strcat(line,t3); strcat(line,"."); strcat(line, temp);}
        else
            strcat(line, t3);
    }
    else
        strcat(line,temp);

    strcat(line,"\"]"; strcat(line,t4); strcat(line,"\"");
    length = 250 - strlen(line);
    for (i = 0; i < length; i++)
        strcat(line, ";
        fseek(fp,Et,0);
        if ((Et = ftell(fp)) != -1L)
            fputs(line, fp);
```
return;

void check_insert_link(FILE *fp, char Oldt2[200], char *Slist[10], int arc_num)
{
    long int gen[50];
    long int *genPtr = gen;
    int j, k;
    int index = 0;
    char Etype[100], t1[200], t2[200], t3[200], t4[200], line[250];

    gen_offset(Oldt2, genPtr, &k);

    for(j=0; j<k; j++)
    {
        fseek(fp, gen[j], 0);
        fgets(line, 250, fp);
        if(sscanf(line, "%[*%]+["*%]+%[*%]+%[*%]+%[*%]+%[*%]+%[*%]+%[*%]+%[*%]+%[*%]+%[*%]"
                    , Etype, t1, t2, t3, t4) == 5)
            return;
    instance(Etype, Slist, arc_num, 1);
    if(flag == 1)
        j = k;
    return;
}

/*******************************************/

/* After determining the position of the function in the tree, further
investigation is needed so as to determine whether any other instances
of it exists. For example: if we are dealing with 1-arg. functions and
the 1-arg. file already contains a->b. [a]->b. [a]->[a]. a->[b] and we
want to insert a->a.
A graphical representation of the file before insertion is:
(a->b)
    /\    
   / \   (a->[b])
   /   /
(a->a)

Now after insertion the graphical representation changes to:
(a->b)
    /\  
   /   ([a]->[b]) (a->a)
   /   /
   (a->[a])
After calling the function insert. The type, (a->a) has (a->b) as its
general types and no specific types. But, after calling insert_final
all functions that are more specific points to it directly or
indirectly. */

void insert_final(FILE *fp, char *Slist[10], int arc_num, long int St,
                  long int off)
{
    long int spec[50];
    long int *specPtr = spec;
    char Etype[100], t1[200], t2[200], t3[200], t4[200], line[250], temp[200];
    int length, i, j, k, l, b;
Program Listing

fseek(fp, offset, 0);
fgets(line, 250, fp);
if (sscanf(line, "%([^"|][^"|]%[^"|][^"| Теперь текст изображения является чистым и не содержит никаких графических или изображительных элементов. Это позволяет успешно обработать его как простую текстовую строку.Результат чтения текста может быть представлен в различных форматах, включая ASCII, Unicode, HTML, XML и т.д. Форматирование текста может быть сохранено в зависимости от требуемого контекста. Учитывая, что текст изображения является чистым и не содержит никаких графических или изображительных элементов, это позволяет успешно обработать его как простую текстовую строку.Результат чтения текста может быть представлен в различных форматах, включая ASCII, Unicode, HTML, XML и т.д. Форматирование текста может быть сохранено в зависимости от требуемого контекста.
void insertion(int arg_num, char *Slist[10], char *name[20], char *Flist[10])
{
    FILE *fp;
    long int St, oldT, Et;
    char line[250], temp[12];
    char *listtemp[10];
    int i, length, new_arg, b, c;
    Et = oldT = 0;
    Exit_flag = 0;
    check_flag = 0;
    ok = 1;
    backOffset = 0;
    SGindex = SSize = 0;

    b = 0;
    while(b < (arg_num - 1))
    {
        listtemp[b++] = (char *)malloc(30 + sizeof(char));

        if ((fp = fopen(filename[arg_num], "r+b")) == NULL)
            exit(1);
        fseek(fp, 0L, 2);
        if ((St = ftell(fp)) != -1L)
            insert(Slist, St, oldT, Et, fp, name, arg_num, Flist);

        if (Exit_flag == 0)
        {
            for(c = 0; c < SGindex; c++)
                insert_final(fp, Slist, arg_num, St, SGoffset[c]);
            strcpy(line, storage[0]);
            strcat(line, ".");
            strcat(line, name);
            strcat(line, ":\n");
            if (SGindex == 0)
                strcat(line, ":
");
            else
            {
                for(i = 0; i < SGindex; i++)
                {
                    sprintf(temp, ":%d", SGoffset[i]);
                    strcat(line, temp);
                    if (i < SGindex - 1)
                        strcat(line, ":
");
                }
                strcat(line, ":
");
            }
            if (SSize == 0)
                strcat(line, ":\n");
            else
            {
                for(i = 0; i < SSize; i++)
                {
                    sprintf(temp, ":%d", SOffset[i]);
                    strcat(line, temp);
                    if (i < SSize - 1)
                        strcat(line, ":
");
                }
                strcat(line, ":\n");
                if (Ft == 0)
                    strcat(line, ":\n");
                else
                {
                    sprintf(temp, ":%d", Ft);
                    strcat(line, temp);
                }
        }
}
Program Listing

```
strcat(line,"\n");
length = 250 * strlen(line);
for (i = 0; i < length; i++)
  strcat(line, " ");
 fseek(fp, St.c, 0);
if ((St = ftell(fp)) != -1L)
  fputs(line, fp);
fclose(fp);
for(i=0;i<value;i++)
  free(storage[i]);

for(i=0;i<arg_num;i++)
{
  new_arg =0;
  for(b = 0; b < arg_num; b++)
  {
    if (b != i)
      strcpy(listtemp[new_arg++], Slist[b]);
  }
  Ft = St; strcpy(name, " ");
  insertion(new_arg, listtemp.name,Slist);
}

} else
{
  fclose(fp);
  for(i=0;i<value;i++)
    free(storage[i]);
}

/*  ***************************************************************************/
/* This function takes as input a string of names (t1) and the type (Etype)   */
/* that corresponds to those functions. Also, a two-dimensional array is    */
/* used as storage for these names and type by constantly updating and      */
/* removing duplicates.*/
void find_names(char Etype[100], char t1[200], char *temp[2][100], int *num)
{
  int i, j, k, l;
  char **nam[50];
  char **namPtr = nam;
  k = *num;
  l = k;

  function_names(t1, namPtr, &i);

  for(j=0;j<i;j++)
    if (multi_memb(nam[j], temp, l) == 0)
    {
      temp[0][k] = (char *)malloc(100*sizeof(char));
      temp[1][k] = (char *)malloc(50*sizeof(char));
      strcpy(temp[0][k],Etype);
      strcpy(temp[1][k],nam[j]);
      k++;
    }

  *num = k;
```
return;
}/* .......................................................... */
/* This function uses the offset stored with the type to obtain all general
types. */
void find_general(char t2[200], FILE *fp) {
    int j,k,z,y,b;
    long int temp[50], general[100], answer[100];
    long *genPtr = general;
    long *tempPtr = temp;
    char Etype[100], t1[200], line[250];

    gen_offset(t2, genPtr, &k);
    while (k > 0) {
        z = 0;
        for(j=0; j<k; j++) {
            fseek(fp, general[j], 0);
            fgets(line, 250, fp);
            sscanf(line, "\%[\'\'][\'\']\%[\'\'][\'\']\%[\'\'][\'\']Etype.t1.t2",
                   gen_offset(t2, tempPtr, &y);
                    for(b=0; b<y; b++)
                        answer[b] = temp[b];
                find_names(Etype.t1, general_functions, &general_index);
        }
        k = z;
        for(j=0; j<z; j++)
            general[j] = answer[j];
    }
    return;
}

/* .......................................................... */
/* This function uses the offset stored with the type to obtain all
specific types. */
void find_specific(char t3[200], FILE *fp) {
    int j,k,z,y,b;
    long int temp[50], spec[100], answer[100];
    long *specPtr = spec;
    long *tempPtr = temp;
    char Etype[100], t1[200], t2[200], line[250];

    gen_offset(t3, specPtr, &k);
    while (k > 0) {
        z = 0;
        for(j=0; j<k; j++) {
            fseek(fp, spec[j], 2);
            fgets(line, 250, fp);
            sscanf(line, "\%[\'\'][\'\']\%[\'\'][\'\']\%[\'\'][\'\']Etype.t1.t2.t3",
                   gen_offset(t3, tempPtr, &y);
                    for(b=0; b<y; b++)
                        answer[b] = temp[b];
                find_names(Etype.t1, specific_functions, &specific_index);
        }
        k = z;
        for(j=0; j<z; j++)
            spec[j] = answer[j];
    }
    return;
Program Listing

```c
answer[z++] = temp[b];
find_names(ETYPE, t1.specific_functions, &specific_index);
}
k = z;
for(j=0;j<k;j++)
    spec[j] = answer[j];

return;
} /* ........................................................................... */
/* This function uses the offset stored with the type to obtain all types 
   that are similar by freezing argument(s). */

void find_freezing(char t4[200], FILE *fp, int arg_num)
{
    int j,k,z,y,b;
    long int temp[50], freeze[100], answer[100];
    long int *freezePtr = freeze;
    long int *tempPtr = temp;
    char Etype[100], t1[200], t2[200], line[250], t3[200];

gen_offset(t4, freezePtr, &k);
z = 0;

for(j=0;j<k;j++)
{
    fseek(fp, freeze[j], 0);
    fgets(line, 250, fp);
    sscanf(line, "%[^{][{][{][{][{][{][{]}.Etype,t1,t2,t3,t4]}
GEN_offset(t2, tempPtr, &y);
    for(b=0;b<y;b++)
        answer[z++] = temp[b];
    find_names(ETYPE, t1.freeze_functions, &freeze_index);
}

k = z;
for(j=0;j<k;j++)
    freeze[j] = answer[j];

while (k > 0)
{
    z = 0;
    for(j=0;j<k;j++)
    {
        fseek(fp, freeze[j], 0);
        fgets(line, 250, fp);
        sscanf(line, "%[^{][{][{][{][{][{][{]}.Etype,t1,t2]}
GEN_offset(t2, tempPtr, &y);
    for(b=0;b<y;b++)
        answer[z++] = temp[b];
    find_names(ETYPE, t1.freeze_functions, &freeze_index);
}
    k = z;
    for(j=0;j<k;j++)
        freeze[j] = answer[j];
}
return;
}
```
/* Opens the required files so as to obtain freeze·argument(s) 
function(s).* /

void freezing(char t4[200], int arg_num) 
{ 
  FILE *fp;

  if (strcmp(t4, "") != 0) 
  { 
    arg_num++;
    fp = fopen(filename[arg_num], "rb");
    find_freezing(t4, fp, arg_num);
    fclose(fp);
  }
  return;
}

/* Searches for an exact match by using instance relation. If the type does not exist then it is inserted. Finally all specific, general and freeze functions are obtained and stored in global arrays. */

void search_type(int count, FILE *fp, char *list[10], long int spec[50],
int arg_num) 
{
  int i,j,k;
  long int *specPtr = spec;
  char line[250];
  char Etype[100], t1[200], t2[200], t3[200], t4[200], temp[200];
  char funct_name[20] = " ";
  char *Flist[10];

  for(i=0;i<count;i++)
  {
    fseek(fp, spec[i], 0);
    fgets(line, 250, fp);
    sscanf(line, "[%’{}][%’{}][%’{}][%’{}][%’{}][%’{}][%’{}][%’{}],Etype,t1,t2,t3,t4);
    instance(Etype, list, arg_num, 0);
    if (flag == 1)
      i = count;
  }
  if (flag ==1) 
  {
    if (exact_match == 1)
    {
      free_storage();
      if (strcmp(t1,"") == 0)
      {
        find_general(t2, fp);
        find_specific(t3, fp);
        freezing(t4, arg_num);
      }
      else
      {
        find_names(Etype, t1, exact_functions, &exact_index);
        find_general(t2, fp);
        find_specific(t3, fp);
        freezing(t4, arg_num);
      }
    }
  }
else
    |
    gen_offset(t3, specPtr, &j);
    if (j == 0)
    |
    free_storage();
    insertion(arg_num, list, funct_name, Flist);
    retrieve(list, arg_num);
    }
    else
    |
    search_type(j, fp, list, spec, arg_num);
    |
    }
else
    |
    free_storage();
    insertion(arg_num, list, funct_name, Flist);
    retrieve(list, arg_num);
    
    return;
|

/* .......................................................... */

void retrieve(char *Slist[10], int arg_num)
|
    FILE *fp;
    long int spec[50];
    int k = 1;
    check_flag = 0;
    exact_index = general_index = specific_index = freeze_index = 0;

    if ((fp = fopen(filename[arg_num], "rb")) == NULL)
        exit(1);

    spec[0] = 0;
    search_type(k, fp, Slist, spec, arg_num);
    fclose(fp);
|

/* .......................................................... */

/* inserts the new record after removing name if it exists. */
void remove_name(char name[20], FILE *fp, char Etype[100], char t1[200],
    char t2[200], char t3[200], char t4[200], long int offset)
|
    int i, b, j, length;
    char *nam[50];
    char **namPtr = nam;
    char line[250];

    strcpy(line, Etype);
    strcat(line, ":");
    function_names(t1, namPtr, i);
    if (member(name, nam, i) == 1)
    |
    if (i == 1)
        strcat(line, ":");
    else

97


```c
{ 
  b= 0;
  for(j=0; j<i; j++)
  if (strcmp(nam[j], name) != 0)
    strcpy(nam[b++], nam[j]);
  for(i=0; i<b; i++)
  { 
    strcat(line, nam[i]);
    if (i < b - 1)
      strcat(line, ",");
  }
  strcat(line, "}"); strcat(line, t2); strcat(line, "}");
  strcat(line, t3); strcat(line, ");"); strcat(line, t4); strcat(line, "}");
  strlen(line);
  for (i = 0; i < strlen(line); i++)
    strcat(line, ",");
  fseek(fp, offset, 0);
  fputs(line, fp); 
  else
    XtManageChild(no_name_del);
    return;
  }

/* .............................................................. */
/* Searches for an exact match so that the required function can be */
/* deleted. */
void delete(int count, FILE *fp, char *list[10], long int spec[50],
            int arg_num, char name[20])
{
  int i, j, k;
  long int *specPtr = spec;
  char line[250];
  char Etype[100], t1[200], t2[200], t3[200], t4[200];

  for(i=0; i<count; i++)
  { 
    fseek(fp, spec[i], 0);
    fgets(line, 250, fp);
    sscanf(line, "+\[\%[\`-][\%\`-]\%\`-][\%\`-]\%\`-\]","*\%\`-\]","*\%\`-\]","*\%\`-\]","*\%\`-\]%\`-\]","%\`-\]",&type, t1, t2, t3, t4);
    instance(Etype, list, arg_num, 0);
    if (flag == 1) { k=i;
      i = count;
    }
  }
  if (flag ==1)
  { 
    if (exact_match == 1)
    { 
      free_storage();
      remove_name(name, fp, Etype, t1, t2, t3, t4, spec[k]);
    }
    else
    { 
      gen_offset(t3, specPtr, &j);
      if (j == 0)
      { 
        free_storage();
      }
  ```
XtManageChild(no_del);
    }
    else
      delete(j, fp, list, spec, arg_num, name);
    }
  }

  else
    free_storage();
  XtManageChild(no_del);
}

return;
}

/* ........................................................................... */

void delete_name(char *list[10], int arg_num, char name[20])
{
    FILE *fp;
    long int spec[20];
    int k = 1;
    check_flag = 0;

    fp = fopen(filename[arg_num], "r+b");
    spec[0] = 0;
    delete(k, fp, list, spec, arg_num, name);

    fclose(fp);
}

/* ........................................................................... */

void browser(char gen[200], char specif[200], char freeze[200], int arg_num,
              FILE *fp)
{
    int i,j,k,m;
    char line[250];
    FILE *filePointer;
    char Etype[200], t1[200], t2[200], t3[200], t4[200], temp[200];
    long int general[50];
    long int *genPtr = general;
    long int spec[50];
    long int *specPtr = spec;
    long int freeze[50];
    long int *freezePtr = freeze;

    gen_offset(specif, specPtr, &i);
    gen_offset(freeze, freezePtr, &k);
    gen_offset(gen, genPtr, &j);

    for(m=0; m<i; m++)
      {
        fseek(fp, spec[m], 0);
        fgets(line, 250, fp);
        sscanf(line, "%[^{][%[^}][{][^}]][^}][%[^}][{][^}]][^}]").Etype, t1, t2, t3, t4);
        strcpy(spec_browser[0] [spec_num_bro], Etype);
        strcpy(spec_browser[1] [spec_num_bro], t1);
        strcpy(spec_browser[2] [spec_num_bro], t2);
        strcpy(spec_browser[3] [spec_num_bro], t3);
        strcpy(spec_browser[4] [spec_num_bro], t4);
        ++spec_num_bro;
      }
for (m=0;m<n;m++)
{
    fseek(fp, general[m],0);
    fgets(line,250,fp);
    sscanf(line, "%"[^"{}[]{}[]["]]%"[^"]]%"", Et, t1,t2,t3,t4);

    strcpy(gen_browser[0][gen_num_bro], Et);
    strcpy(gen_browser[1][gen_num_bro], t2);
    strcpy(gen_browser[2][gen_num_bro], t3);
    strcpy(gen_browser[3][gen_num_bro], t4);
    ++gen_num_bro;
}

if (k !=0)
    filePointer = fopen(filename[+arg_num], "rb");
for (m=0;m<k;m++)
{
    fseek(filePointer, freeze[m],0);
    fgets(line,250,filePointer);
    sscanf(line, "%"[^"{}[]{}[]["]]%"[^"]]%"", Et, t1,t2,t3,t4);

    strcpy(freeze_browser[0][free_num_bro], Et);
    strcpy(freeze_browser[1][free_num_bro], t2);
    strcpy(freeze_browser[2][free_num_bro], t3);
    strcpy(freeze_browser[3][free_num_bro], t4);
    ++free_num_bro;
}

if (k !=0)
    fclose(filePointer);

/]  ----------------------------------------------- */

/*  --------------- USER INTERFACE SOURCE CODE WRITTEN IN MOTIF  
   ----------------------------------------------- */

void GetColor(Widget widget)
{
    XColor color1, unused;
    Colormap cmap = DefaultColormapOfScreen(XtScreen(widget));

    XAllocNamedColor(XtDisplay(widget), cmap, "red", &Red, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "blue", &Blue, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "green", &Green, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "yellow", &Yellow, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "pink", &Pink, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "violet", &Violet, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "black", &Black, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "white", &White, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "Navy", &Navy, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "Orange", &Orange, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "Magenta", &Magenta, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "Cyan", &Cyan, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "Brown", &Brown, &unused);
    XAllocNamedColor(XtDisplay(widget), cmap, "grey", &Grey, &unused);

    XColor color2;
    color2 = colormapAllocate(cmap, 1, 0x00000000);
    cmap = colormapGetReferenced(cmap);
    colormapDestroy(cmap);
    colormapDestroy(color1);
    colormapDestroy(color2);
    colormapDestroy(unused);
}
XAllocNamedColor(XtDisplay(widget), cmap, "LimeGreen", &LimeGreen, &unused);
XAllocNamedColor(XtDisplay(widget), cmap, "Turquoise", &Turquoise, &unused);
XAllocNamedColor(XtDisplay(widget), cmap, "Wheat", &Wheat, &unused);
XAllocNamedColor(XtDisplay(widget), cmap, "Purple", &Purple, &unused);

}  /* This is used to reset widgets associated with the insert, delete & retrieve popup window. */
void resetdialog(Widget list, Widget scale, Widget text, Widget text1)
{
  Arg ar[10];
  int ac;

  argument[0] = "Argument1";
  argument[1] = "Argument2";
  argument[2] = "Argument3";
  argument[3] = "Argument4";
  argument[4] = "Argument5";
  argument[5] = "Argument6";
  argument[6] = "Argument7";
  argument[7] = "Argument8";
  strcpy(result, "");
  val = 0;
  c = 0;

  XmListDeleteAllItems(list);
  XtSetSensitive(list, False);

  ac = 0;
  XtSetArg(ar[ac], XmNvalue, 0); ac++;
  XtSetValues(scale, ar, ac);

  ac = 0;
  XtSetArg(ar[ac], XmNvalue, ""); ac++;
  XtSetValues(text, ar, ac);
  XtSetSensitive(text, False);

  ac = 0;
  XtSetArg(ar[ac], XmNvalue, ""); ac++;
  XtSetValues(text1, ar, ac);
}

/*************************************************************************/
void reset_text(Widget text)
{
  Arg ar[10];
  int ac;

  ac = 0;
  XtSetArg(ar[ac], XmNvalue, ""); ac++;
  XtSetValues(text, ar, ac);
}

/*************************************************************************/
void buttonCB(w, client_data, call_data)

Widget w;
int client_data;
XmPushButtonCallbackStruct *call_data;
/* Program Listing */

char *s;
Arg al[10];
int ac;
char k[100] = "";
char j[6] = " -> ";
switch(client_data)
{
    case APPLY:
        s = XmTextGetString(text1);
        strcpy(result, s);
        for (ac = 0; ac < val; ac++)
        {
            strcat(k, argument[ac]);
            strcat(k, j);
        }
        strcat(k, result);
        XtManageChild(dialog1);
        ac = 0;
        XtSetArg(al[ac], XmNmessageString,
            XmStringCreateLocale(k, char_set)); ac++;
        XtSetValues(dialog1, al, ac);
        XtUnmanageChild(bb_dialog);
        XtFree(s);
        break;
    case QUIT:
        XtUnmanageChild(bb_dialog);
        break;
    case RESET:
        resetdialog(list, scale, text, text1);
        break;
    case HELP:
        XtManageChild(help_ret);
        break;
}
*/

void buttonCB_insert(w, client_data, call_data)
Widget w;
int client_data;
XmPushButtonCallbackStruct *call_data;
{
    char *s;
    Arg al[10];
    int ac;
    char k[100] = "";
    char j[6] = " -> ";
    switch(client_data)
    {
    case APPLY:
        s = XmTextGetString(text1);
        strcpy(result, s);
        s = XmTextGetString(text4);
        strcpy(func_name, s);
        for (ac = 0; ac < val; ac++)
        {
            strcat(k, argument[ac]);
            strcat(k, j);
        }
        strcat(k, result);
        XtManageChild(dialog1);
        ac = 0;
        XtSetArg(al[ac], XmNmessageString,
            XmStringCreateLocale(k, char_set)); ac++;
        XtSetValues(dialog1, al, ac);
        XtUnmanageChild(bb_dialog);
        XtFree(s);
        break;
Program Listing

```c
}  
strcat(k.result): strcat(k," ");
strcat(k.funct_name): strcat(k,"|" );
XtManageChild(dial);
ac = 0;
XtSetArg(al[ac], XmNmessageString,
    XmStringCreateToR(k,char_set)); ac++;
XtSetValues(dial, al, ac);
XtUnmanageChild(bb_dia);
XtFree(s);
break;
case QUIT:
    XtUnmanageChild(bb_dia);
bkreak:
case RESET:
    redialog(lis,scal, tex, tex1);
    reset_text (text4);
    break;
case HELP:
    XtManageChild (help_inc);
    break;

}  

/*---------------------------------------------------------------------------*/

void buttonCB_delete(w, client_data, call_data)

    Widget w;
    int client_data;
    XmPushButtonCallbackStruct *call_data;
{
    char *s;
    Arg ai[10];
    int ac;
    char k[100] = " ";
    char j[10] = " ";
    switch (client_data)
    {
        case APPLY:
            s = XmTextGetString(tel);
            strcpy(result, s);
            s = XmTextGetString(text4);
            strcpy(funct_name, s);
            for (ac = 0; ac < val; ac++)
            {
                strcat(k,argument[ac]);
                strcat(k, j);
            }
            strcat(k, result); strcat(k," ");
            strcat(k, funct_name); strcat(k," | " );
            XtManageChild(dial_del);
            ac = 0;
            XtSetArg(al[ac], XmNmessageString,
            XmStringCreateToR(k,char_set)); ac++;
            XtSetValues(dial_del, al, ac);
            XtUnmanageChild(bb_dia_del);
            XtFree(s);

            break;
        case QUIT:
```
XtUnmanageChild(bb_dialog_del);
    break;
  case RESET:
    resetdialog(li,sca,te,tel);
    reset_text(tex1);
    break;
  case HELP:
    XtManageChild(help_del);
    break;
  }
  /* ...........................................................*/

void add_items(Widget list)
{
    XmString s;
    int list_cnt;

    XtSetSensitive(list,True);
    XmListDeleteAllItems(list);
    for (list_cnt = 0; list_cnt < vnl; list_cnt++)
    {
        s=XmStringCreate(argument[list_cnt],char_set);
        XmListAddItem(list,s,0);
        XmStringFree(s);
    }
  }/* ...........................................................*/

void handle_scale(w,client_data,call_data)
    Widget w;
    XtPointer client_data;
    XmScaleCallbackStruct *call_data;
    {
        val = call_data->value;
        if (val == 0)
        {
            XtSetSensitive(text,False);
            XtSetSensitive(list,False);
        }
        else
        {
            add_items(list);
        }
    }/* ...........................................................*/

void handle_scale_insert(w,client_data,call_data)
    Widget w;
    XtPointer client_data;
    XmScaleCallbackStruct *call_data;
    {
        val = call_data->value;
        if (val == 0)
        {
            XtSetSensitive(text,False);
            XtSetSensitive(list,False);
        }
        else
        {
            add_items(list);
        }
    }/* ...........................................................*/

void handle_scale_delete(w,client_data,call_data)

Widget w;
XtPointer client_data;
XmScaleCallbackStruct *call_data;
{
    val = call_data->value;
    if (val == 0)
        XtSetSensitive(te,False);
    else
        XtSensitive(li,False);
}
void selectCB(w,client_data,call_data)
/*....................................................................*/
Widget w;
XtPointer client_data;
XmAnyCallbackStruct *call_data;
{
    Arg al[10];
    int ac;
    int *pos_list;
    int pos_list_length;
    int x, *p;
    int mem_allocated;

    mem_allocated = XmListGetSelectedPos(list,&pos_list,
        &pos_list_length);
    p = pos_list;
    c = -1;
    for(x=0;x<pos_list_length;x++)
        c = *p++;

    if (c > 0)
        
        XtSetSensitive(text,True);
        ac=0;
        XtSetArg(al[ac],XmNValue,argument[ac--];
        XtSetArg(al[ac],XmNCursorPosition,strlen(argument[ac]);
            ac++;
        XtSetValues(text,al,ac);
    }
    if (mem_allocated)
        XtFree(pos_list);
    /*....................................................................*/
void selectCB_insert(w,client_data,call_data)
/*....................................................................*/
mem_allocated = XmListGetSelectedPos(lis,&pos_list, &pos_list_length);
p = pos_list;
c = 3;
for(x=0;x<pos_list_length;x++)
c = *p++;

if (c > 0)
{
    XtSetSensitive(tex,True);
    ac=0;
    XtSetArg(al[ac],XmNvalue,argument[ac]);ac++;
    XtSetArg(al[ac],XmNcursorPosition,strlen(argument[ac]));
    ac++;
    XtSetValues(tex,al,ac);
}

if (mem_allocated)
    XtFree(pos_list);

/*************************************************************************/
void selectCB_delete(w,client_data,call_data)
    Widget w;
    XtPointer client_data;
    XmAnyCallbackStruct *call_data;
{
    Arg al[10];
    int ac;
    int *pos_list;
    int pos_list_length;
    int x, *p;
    int mem_allocated;

    mem_allocated = XmListGetSelectedPos(li,&pos_list, &pos_list_length);
p = pos_list;
c = 3;
for(x=0;x<pos_list_length;x++)
c = *p++;

if (c > 0)
{
    XtSetSensitive(te,True);
    ac=0;
    XtSetArg(al[ac],XmNvalue,argument[ac]);ac++;
    XtSetArg(al[ac],XmNcursorPosition,strlen(argument[ac]));
    ac++;
    XtSetValues(te,al,ac);
}

if (mem_allocated)
    XtFree(pos_list);

/*************************************************************************/

void solutionCB(w,client_data,call_data)
    Widget w;
    int client_data;
    XmAnyCallbackStruct *call_data;

Arg a1[10];
int ac;
int *pos_list;
int pos_list_length;
int x, *q;
int mem_allocated;
FILE *fp;
char line[80];
XmTextPosition position;

mem_allocated = XmListGetSelectedPos(w, &pos_list,
   &pos_list_length);

q = pos_list;
++c;

for(x=0;x<pos_list_length;x++)
c = *q++;

...;
switch(client_data)
|
  case GENERAL:
    fp = fopen(general_functions[1][c],"r");
    break;
  case FREEZE:
    fp = fopen(freeze_functions[1][c],"r");
    break;
  case SPECIFIC:
    fp = fopen(specific_functions[1][c],"r");
    break;
  case EXACT:
    fp = fopen(exact_functions[1][c],"r");
    break;
|
XtSetSensitive(save_item,True);
XtSetSensitive(empty_item,True);
XtSetSensitive(text2,True);
ac = 0;
XtSetArg(a1[ac],XmValue,"");ac++;
XtSetValues(text2,a1,ac);

while (fgets(line,80,fp) != NULL)
|
  position = XmTextGetPosition(text2);
  XmTextInsert(text2,position,line);
|
fclose(fp);

if (mem_allocated)
  XtFree(pos_list);
|
/**...................................................................................
 void add_sol(Widget wid, char *templist[2][100], int count)
 |
  XmString s;
  int list_cnt:
char stlist[200];
int p,i;

XmListDeleteAllItems(wid);
for (list_cnt = 0; list_cnt < count; list_cnt++)
{
    p = 55 - strlen(templist[0][list_cnt]) - strlen(templist[1][list_cnt]);
    strcpy(stlist, templist[0][list_cnt]);
    for (i = 0; i < p; i++)
        strcat(stlist," ");strcat(stlist," ");
    strcat(stlist, templist[1][list_cnt]);
    s=XmStringCreate(stlist, char_set);
    XmListAddItem(wid, s, 0);
    XmStringFree(s);
}

/*--------------------------------------------------------------------------------*/
void TextCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XmAnyCallbackStruct *call_data;
{
    Arg al[10];
    int ac;
    char *s;

    s = XmTextGetString(text);

    ac=0;
    XtSetArg(al[ac], XmNValue,"");ac++;
    XtSetValues(text, al, ac);
    XtSetSensitive(text, True);
    argument[c] = s;
    add_items(list);
}

/*--------------------------------------------------------------------------------*/
void TextCB_insert(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XmAnyCallbackStruct *call_data;
{
    Arg al[10];
    int ac;
    char *s;

    s = XmTextGetString(text);

    ac=0;
    XtSetArg(al[ac], XmNValue,"");ac++;
    XtSetValues(text, al, ac);
    XtSetSensitive(text, False);
    argument[c] = s;
    add_items(list);
}

/*--------------------------------------------------------------------------------*/
void TextCB_delete(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XmAnyCallbackStruct *call_data;
{ 
    Arg a1[10];
    int ac;
    char *s;

    s = XmTextGetString(te);

    ac=0;
    XtSetArg(a1[ac], XmNValue, "") ; ac++;
    XtSetValues(te, a1, ac);
    XtSetSensitive(te, False);
    argument[c] = s;
    add_items(li);
}

void placeWidgets()
{
    Arg a1[20];
    int ac;

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 1); ac++;
    XtSetValues(frame1, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 115); ac++;
    XtSetValues(sep4, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 120); ac++;
    XtSetValues(scale, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 195); ac++;
    XtSetValues(sep1, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 195); ac++;
    XtSetValues(label2, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 100); ac++;
    XtSetArg(a1[ac], XmNY, 195); ac++;
    XtSetValues(frame, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 245); ac++;
    XtSetValues(sep2, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNX, 1); ac++;
    XtSetArg(a1[ac], XmNY, 255); ac++;
    XtSetValues(label1, a1, ac):
}

ac = 0;
 XtSetArg(al[ac],XmNX,100);ac++;
 XtSetArg(al[ac],XmNY,255);ac++;
 XtSetValues(frame2, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,310);ac++;
 XtSetValues(sep3, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,320);ac++;
 XtSetValues(rowcol, al,ac);
}

/*****************************************************************************/
void placeWidgets_insert()
{
 Arg al[20];
 int ac;

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,1);ac++;
 XtSetValues(f1, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,115);ac++;
 XtSetValues(s4, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,120);ac++;
 XtSetValues(scal, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,185);ac++;
 XtSetValues(s1, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,195);ac++;
 XtSetValues(l2, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,100);ac++;
 XtSetArg(al[ac],XmNY,195);ac++;
 XtSetValues(f, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,245);ac++;
 XtSetValues(s2, al,ac);

ac = 0;
 XtSetArg(al[ac],XmNX,1);ac++;
 XtSetArg(al[ac],XmNY,255);ac++;
XtSetValues(l1, al.ac);

ac = 0;
   XtSetArg(al[ac], XmNX, 100); ac++;
   XtSetArg(al[ac], XmNY, 255); ac++;
   XtSetValues(f2, al, ac);

ac = 0;
   XtSetArg(al[ac], XmNX, 1); ac++;
   XtSetArg(al[ac], XmNY, 310); ac++;
   XtSetValues(s3, al, ac);

ac = 0;
   XtSetArg(al[ac], XmNX, 1); ac++;
   XtSetArg(al[ac], XmNY, 320); ac++;
   XtSetValues(l4, al, ac);

ac = 0;
   XtSetArg(al[ac], XmNX, 100); ac++;
   XtSetArg(al[ac], XmNY, 320); ac++;
   XtSetValues(f4, al, ac);

ac = 0;
   XtSetArg(al[ac], XmNX, 1); ac++;
   XtSetArg(al[ac], XmNY, 375); ac++;
   XtSetValues(roco, al, ac);

void placeWidgets_delete()
{
   Arq al[20];
   int ac;

   ac = 0;
      XtSetArg(al[ac], XmNX, 1); ac++;
      XtSetArg(al[ac], XmNY, 1); ac++;
      XtSetValues(fri, al, ac);

   ac = 0;
      XtSetArg(al[ac], XmNX, 1); ac++;
      XtSetArg(al[ac], XmNY, 115); ac++;
      XtSetValues(se4, al, ac);

   ac = 0;
      XtSetArg(al[ac], XmNX, 1); ac++;
      XtSetArg(al[ac], XmNY, 120); ac++;
      XtSetValues(sca, al, ac);

   ac = 0;
      XtSetArg(al[ac], XmNX, 1); ac++;
      XtSetArg(al[ac], XmNY, 185); ac++;
      XtSetValues(sel, al, ac);

   ac = 0;
      XtSetArg(al[ac], XmNX, 1); ac++;
      XtSetArg(al[ac], XmNY, 195); ac++;
      XtSetValues(la2, al, ac);

   ac = 0;
XtSetArg(al[ac], XmNX, 100); ac++;
XtSetArg(al[ac], XmNY, 195); ac++;
XtSetValues(fr, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 1); ac++;
XtSetArg(al[ac], XmNY, 245); ac++;
XtSetValues(se2, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 1); ac++;
XtSetArg(al[ac], XmNY, 255); ac++;
XtSetValues(la1, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 100); ac++;
XtSetArg(al[ac], XmNY, 255); ac++;
XtSetValues(fr2, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 1); ac++;
XtSetArg(al[ac], XmNY, 310); ac++;
XtSetValues(se3, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 1); ac++;
XtSetArg(al[ac], XmNY, 320); ac++;
XtSetValues(la2, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 100); ac++;
XtSetArg(al[ac], XmNY, 320); ac++;
XtSetValues(fr4, al, ac);
ac = 0;
XtSetArg(al[ac], XmNX, 1); ac++;
XtSetArg(al[ac], XmNY, 375); ac++;
XtSetValues(rc, al, ac);
}

/* ......................................................*/

void dialogCB(w, client_data, call_data)
  Widget w;
  int client_data;
  XmAnyCallbackStruct *call_data;
  {
    Arg al[20];
    int ac:

    switch(client_data)
    {
      case GENERAL:
        XtManageChild(dialog); ac = 0;
        XtSetArg(al[ac], XmNX, 200); ac++;
        XtSetArg(al[ac], XmNY, 200); ac++;
        XtSetValues(dialog, al, ac);
        if (general_index == 0)
          XtManageChild(dialog2):
        else
          

Blackford
XtManageChild(bb_dialog1); ac = 0;
XtSetArg(al[ac], XmNx, 250); ac++;
XtSetArg(al[ac], XmNy, 200); ac++;
XtSetValues(bb_dialog1, al, ac);
add_sol(list1, general_functions, general_index);
}
if (freeze_index == 0)
XtManageChild(dial2);
else
{
XtManageChild(bb_dialog1);
add_sol(list1, freeze_functions, freeze_index);
}
break;
case SPECIFIC:
if (specific_index == 0)
XtManageChild(dial2);
else
{
XtManageChild(bb_dialog1);
add_sol(list1, specific_functions, specific_index);
}
break;
case QUIT:
XtUnmanageChild(w);
bvreak;
}

/* .......................................................... */

void dialog2CB(w, client_data, call_data)
Widget w;
int client_data;
XmAnyCallbackStruct *call_data;
{
Arg al[20];
int ac;

switch(client_data)
{
case GENERAL:
if (general_index == 0)
XtManageChild(dialog3);
else
{
XtManageChild(bb_dialog1); ac = 0;
XtSetArg(al[ac], XmNx, 250); ac++;
XtSetArg(al[ac], XmNy, 200); ac++;
XtSetValues(bb_dialog1, al, ac);
add_sol(list1, general_functions, general_index);
}
if (freeze_index == 0)
XtManageChild(dial2);
else
{
XtManageChild(bb_dialog1);
add_sol(list1, freeze_functions, freeze_index);
}
XtSetSensitive(butt[0], True);
bvreak;
case SPECIFIC:
    if (specific_index == 0)
        XtManageChild(dia2);
    else
        {  
            XtManageChild(bb_dial);
            add_scl(lst1, specific_functions, specific_index);
        }
    break;
    case QUIT:
        XtUnmanageChild(bb_dialog2);
        break;
    }
}/*---------------------------------------------*/
void dialog3CB (w, client_data, call_data)
    Widget w;
    int client_data;
    XmAnyCallbackStruct *call_data;
    {
        switch(client_data)
        {
        case GENERAL:
            XtUnmanageChild(bb_dialog1);
            break;
        case FREEZE:
            XtUnmanageChild(bb_dial1);
            break;
        case SPECIFIC:
            XtUnmanageChild(bb_dial);
            break;
        }
    }/*---------------------------------------------*/
void dialog1CB (w, client_data, call_data)
    Widget w;
    int client_data;
    XmAnyCallbackStruct *call_data;
    {
        Arg a1[10];
        int ac, k;

        switch(client_data)
        {
        case YES:
            retrieve(argument, val);
            if (exact_index == 0)
                { 
                    XtManageChild(dialog); ac = 0;
                    XtSetArg(a1[ac], XmNx, 200); ac++;
                    XtSetArg(a1[ac], XmNy, 200); ac++;
                    XtSetValues(dialog, a1, ac);
                }
            else
                {  
                    XtManageChild(bb_dialog2);
                    add_scl(lst3, exact_functions, exact_index);
                }
            XtUnmanageChild(dialog1);
break;
case NO:
    XtManageChild(bb_dialog);
    XtUnmanageChild(dial);
    break;
}
XtUnmanageChild(w);
}/*-----------------------------------------------*/

void dialog1CB_insert(w, client_data, call_data)

Widget w;
int client_data;
XmAnyCallbackStruct *call_data;

Arg al[10];
int ac, k;
char *Flist[8];

switch (client_data)
{
case YES:
    insertion(val, argument, funct_name, Flist);
    break;
case NO:
    XtManageChild(bb_dia);
    XtUnmanageChild(dial);
    break;
}
XtUnmanageChild(w);
}/*-----------------------------------------------*/

void dialog1CB_delete(w, client_data, call_data)

Widget w;
int client_data;
XmAnyCallbackStruct *call_data;

Arg al[10];
int ac, k;

switch (client_data)
{
case NAME :
    delete_name(argument, val, funct_name);
    break;
case RECORD:
    XtManageChild(dial_del);
    break;
case NO :
    XtManageChild(bb_dia_del);
    XtUnmanageChild(dial_del);
    break;
}
}/*-----------------------------------------------*/
int ac:
Arg al[10];

ac = 0;
XtSetArg(al[ac], XmNacceleratorText,
   XmStringCreate(acc_text, XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac], XmNaccelerator, key); ac++;
XtSetValues(w, al, ac);

/*==================================================================*/
void setup_gc()
{
    int foreground, background;
    XGCValues vals;
    Arg al[10];
    int ac;
    XFontStruct *font=NULL;
    char *SMAPoketFont=NULL;

    SMALLFONT = "6x10";
    font=XLoadQueryFont(XDisplay(drawing_area), SMALLFONT);

    ac=0;
    XtSetArg(al[ac], XmNforeground, &foreground); ac++;
    XtSetArg(al[ac], XmNbackground, &background); ac++;
    XtGetValues(drawing_area, al, ac);

    vals.foreground = foreground;
    vals.background = background;
    vals.font = font->fid;
    gc = XtGetGC(drawing_area, GCForeground | GCForeground
        | GCBackground
        | GCFont, &vals);

    vals.foreground = foreground;
    vals.background = background;
    vals.line_width = 4;
    ngc = XtGetGC(drawing_area, GCForeground | GCForeground
        | GCLineWidth, &vals);
}

void place_type()
{
    int m, j;
    int x, y;
    char temp[200];

    XDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
        gc, 105, 150, type_browser, strlen(type_browser));
    XDrawRectangle(XDisplay(drawing_area),
        XtWindow(drawing_area), gc, 100, 240,
        (6*strlen(type_browser)+6), 15);

    y = 280; x = 2;
    for(m=2:2<spec_num_bro;m++)
    {
        if (x + (6*strlen(spec_browser[0][m])) < 550)
        {
```
xDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
    gc,x+3,y+10,spec_browser[0][m].strlen(spec_browser[0][m]));
xDrawRectangle(XDisplay(drawing_area),
    XtWindow(drawing_area), gc, x, y,
    (6*strlen(spec_browser[0][m])+6),15);
spec_bro[0][m] = x;
spec_bro[1][m] = y;
x+=6*strlen(spec_browser[0][m])+12;
}
else
{
x = 2; y+=20;
xDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
    gc,x+3,y+10,spec_browser[0][m].strlen(spec_browser[0][m]));
xDrawRectangle(XDisplay(drawing_area),
    XtWindow(drawing_area), gc, x, y,
    (6*strlen(spec_browser[0][m])+6),15);
spec_bro[0][m] = x;
spec_bro[1][m] = y;
x+=6*strlen(spec_browser[0][m])+12;
}
}

y = 20; x = 2;
for(m=0;m<gen_num_bro;m++)
{
    if ((x + (6*strlen(gen_browser[0][m])+6)) < 550)
    {
        xDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
            gc,x+3,y+10,gen_browser[0][m].strlen(gen_browser[0][m]));
xDrawRectangle(XDisplay(drawing_area),
            XtWindow(drawing_area), gc, x, y,
            (6*strlen(gen_browser[0][m])+6),15);
gen_bro[0][m] = x;
gen_bro[1][m] = y;
x+=6*strlen(gen_browser[0][m])+12;
    }
    else
    {
        x = 2; y+=20;
xDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
            gc,x+3,y+10,gen_browser[0][m].strlen(gen_browser[0][m]));
xDrawRectangle(XDisplay(drawing_area),
            XtWindow(drawing_area), gc, x, y,
            (6*strlen(gen_browser[0][m])+6),15);
gen_bro[0][m] = x;
gen_bro[1][m] = y;
x+=6*strlen(gen_browser[0][m])+12;
    }
}

y = 20; x = 560;
for(m=0;m<free_num_bro;m++)
{
    xDrawString(XDisplay(drawing_area), XtWindow(drawing_area),
        gc,x+3,y+10,freeze_browser[0][m],
        strlen(freeze_browser[0][m]));
xDrawRectangle(XDisplay(drawing_area),
        XtWindow(drawing_area), gc, x, y,
```
(6*strlen(freeze_browser[0][m])+6), 15);
freeze_bro[0][m] = x;
freeze_bro[1][m] = y;
y+=20;
}

XDrawLine(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 550, 0, 550, 500);
XDrawLine(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 0, 230, 550, 230);
XDrawLine(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 0, 260, 550, 260);
XDrawString(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 1, 15, "PARENT LEVEL", 12);
XDrawString(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 1.275, "CHILDRENLEVEL", 14);
XDrawString(XtDisplay(drawing_area),
    XtWindow(drawing_area), ngec, 560, 15, "FREEZE LEVEL", 12);
}

int type_pos(int list[2][20], int num, std a[4][20], int xpos, int ypos)
{
    int m;

    for(m=0; m<num; m++)
    {
        if ((xpos >= list[0][m])
            && (xpos <= (list[0][m] + 6*strlen(a[0][m])+6))
            && (ypos >= list[1][m]) && (ypos <= (15 + list[1][m])))
            return m;
        
    }
    return -1;
}

void handle_click(w, client_data, event)
{
    Widget w;
    XtPointer client_data;
    XEvent *event;

    int temp;
    FILE *fp;

    temp = type_pos(freeze_bro, free_num_bro, freeze_browser,
                    event->xbutton.x, event->xbutton.y);
    if (event->xbutton.x > 550)
    {
        if (temp > -1)
        {
            ++browse_arg;

            fp = fopen(filename[browse_arg], "rb");
            strcpy(type_browser, freeze_browser[0][temp]);
            strcat(type_browser, "*");
            strcat(type_browser, freeze_browser[4][temp]);
            strcat(type_browser, ");
            gen_num_bro = spec_num_bro = free_num_bro = 0;
            browser(freeze_browser[1][temp], freeze_browser[2][temp],
                    freeze_browser[3][temp], browse_arg, fp);
            XClearArea(XtDisplay(w), XtWindow(w), 0, 0, 0, True);
        }
    }
}
void exposureCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
    { place_type();
    }

void draw_browser()
    { FILE *fp;
    long int offset = 0;
    char line[250];
char Etype[100], t1[200], t2[200], t3[200], t4[200];

fp = fopen(filename[browse_arg], "rb");
fwrite(fp, offset, 0);
fgets(line, 250, fp);
sscanf(line, "%[^{}]{[^{}]}{[^{}]}{[^{}]}{[^{}]}{[^{}]}}, Etype, t1, t2, t3, t4);
strcpy(type_browser, Etype);
strcat(type_browser, "[");
strcat(type_browser, t1);
strcat(type_browser, "]");
gen_num_bro = spec_num_bro = free_num_bro = 0;
browser(t2, t3, t4, browse_arg, fp);
XtManageChild(form_drawer);
fclose(fp);
}

void BrowCB(w, client_data, call_data)
Widget w;
int client_data;
XmAnyCallbackStruct *call_data;
{
switch (client_data)
{
  case QUIT :
    XtUnmanageChild(form_drawer);
    break;
  case APPLY :
    XtUnmanageChild(form_drawer);
    XtManageChild(bb_browser);
    break;
}
}

void setup_drawA()
{
  Arg al[10];
  int ac;

  ac = 0;
  XtSetArg(al[ac], XmNwidth, 800); ac++;
  XtSetArg(al[ac], XmNheight, 500); ac++;
  form_drawer = XmCreateFormDialog(toplevel, "BROWSER", al, ac);

  ac = 0;
  XtSetArg(al[ac], XmNpacking, XmPACK_COLUMN); ac++;
  XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++;
  XtSetArg(al[ac], XmNnumColumns, 1); ac++;
  XtSetArg(al[ac], XmNpadJustLast, False); ac++;
  XtSetArg(al[ac], XmNspacing, 300); ac++;
  XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
  rowcol_browser =
    XmCreateRowColumn(form_drawer, "rowcol_browser", al, ac);
  XtManageChild(rowcol_browser);
}
Program Listing

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Argument", char_set)); ac++;
buttBrowser[0] =
XmCreatePushButton(rowcol_browser, "buttBrowser0", al, ac);
XtManageChild(buttBrowser[0]);
XtAddCallback(buttBrowser[0], XmNactivateCallback, BrowCB, APPLY);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Quit", char_set)); ac++;
buttBrowser[1] =
XmCreatePushButton(rowcol_browser, "buttBrowser1", al, ac);
XtManageChild(buttBrowser[1]);
XtAddCallback(buttBrowser[1], XmNactivateCallback, BrowCB, QUIT);

ac = 0;
XtSetArg(al[ac], XmNwidth, 790); ac++;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
XtSetArg(al[ac], XmNtopWidget, rowcol_browser); ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
bsep = XmCreateSeparator(form_drawer, "bsep", al, ac);
XtManageChild(bsep);

ac = 0;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
XtSetArg(al[ac], XmNtopWidget, bsep); ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_FORM); ac++;
drawing_area = XmCreateDrawingArea(form_drawer, "drawing_area", al, ac);
XtManageChild(drawing_area);
XtAddCallback(drawing_area, XmNexposeCallback, exposureCB, NULL);
XtAddEventHandler(drawing_area, ButtonReleaseMask,
               FALSE, handle_click, NULL);
setup_gc() ;
}

void browse_scaleCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XmScaleCallbackStruct *call_data;
{
    browse_arg = call_data->value;
}

void setup_browser()
{
    Arg al[10]:

int ac;

ac=0;
XtSetArg(al[ac], XmNwidth, 300); ac++;
XtSetArg(al[ac], XmNheight, 100); ac++;
bb_browser = XmCreateFormDialog(toplevel, "ARG._BROWSER", al, ac);

ac = 0;
XtSetArg(al[ac], XmNtitleString,
    XmStringCreate("Number Of Argument", char_set)); ac++;
XtSetArg(al[ac], XmNoorientation, XmHORIZONTAL); ac++;
XtSetArg(al[ac], XmNshowValue, True); ac++;
XtSetArg(al[ac], XmNminimum, 0); ac++;
XtSetArg(al[ac], XmNmaximum, 8); ac++;
XtSetArg(al[ac], XmNheight, 60); ac++;
XtSetArg(al[ac], XmNwidth, 280); ac++;

XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
scale_browser = XmCreateScale(bb_browser, "ARG._BROWSER", al, ac);
XManageChild(scale_browser);
XtAddCallback(scale_browser, XmNvalueChangedCallback,
    browse_scaleCB, NULL);

ac = 0;
XtSetArg(al[ac], XmNwidth, 290); ac++;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
XtSetArg(al[ac], XmNtopWidget, scale_browser); ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
sep_bro = XmCreateSeparator(bb_browser, "sep_bro", al, ac);
XManageChild(sep_bro);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
    XmStringCreate("OK", char_set)); ac++;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
XtSetArg(al[ac], XmNtopWidget, sep_bro); ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_FORM); ac++;
button_browser =
    XmCreatePushButton(bb_browser, "button_browser", al, ac);
XManageChild(button_browser);
XtAddCallback(button_browser, XmNactivateCallback, draw_browser, CK);
}

/*************************************************************/
void make_menu_label(item_name, menu)
    char *item_name;
    Widget menu;
{
    int ac;
    Arg al[10];

    ac = 0;
    XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR(item_name, XmSTRING_DEFAULT_CHARSET));

122
ac--; 
 XtManageChild(XmCreateLabel(menu, item_name, al, ac));
 }
} /* \ ----------------------------------------------------------*/

void make_menu_separator(menu)
 Widget menu;
 {
 XtManageChild(XmCreateSeparator(menu, "Sep", NULL, 0));
 }

void menuCB(w, client_data, call_data)
 Widget w;
 char *client_data;
 XmAnyCallbackStruct *call_data;
 {
 int ac, control;
 Arg al[10];

 if (strcmp(client_data, "Quit") == 0)
 control = 1;
 else
 if (strcmp(client_data, "Retrieve") == 0)
 control = 2;
 else
 if (strcmp(client_data, "Empty") == 0)
 control = 3;
 else
 if (strcmp(client_data, "Insert") == 0)
 control = 4;
 else
 if (strcmp(client_data, "Delete") == 0)
 control = 5;
 else
 if (strcmp(client_data, "Version") == 0)
 control = 6;
 else
 if (strcmp(client_data, "About") == 0)
 control = 7;
 else
 if (strcmp(client_data, "Help") == 0)
 control = 8;
 else
 control = 9;

 switch (control) {
 case 1:
 exit(0);
 break;
 case 2:
 ac = 0; ft = 0;
 XtManageChild(bb_dialog);
 XtSetArg(al[ac], XmNheight, 200); ac--;
 XtSetArg(al[ac], XmNwidth, 200); ac--;
 XtSetValues(bb_dialog, al, ac);
 break;
 case 3:
 ac = 0;
}
XtSetArg(al[ac]. XmNValue,"" ); ac--;  
XtSetValues(text2, al, ac);  
XtSensitive(save_item,True);  
XtSensitive(empty_item,False);  
XtSensitive(text2,False);  
break;  
case 4 :  
ac=0; Ft = 0;  
XtManageChild(bb_dia);  
XtSetArg(al[ac]. XmNx.150); ac--;  
XtSetArg(al[ac]. XmNy.150); ac--;  
XtSetValues(bb_dia, al, ac);  
break;  
case 5 :  
ac=0;  
XtManageChild(bb_dialog_del);  
XtSetArg(al[ac]. XmNx.300); ac--;  
XtSetArg(al[ac]. XmNy.200); ac--;  
XtSetValues(bb_dialog_del, al, ac);  
break;  
case 6 :  
ac=0;  
XtManageChild(ver);  
XtSetArg(al[ac]. XmNx.300); ac++;  
XtSetArg(al[ac]. XmNy.100); ac--;  
XtSetValues(ver, al, ac);  
break;  
case 7 :  
ac=0;  
XtManageChild(ab_help);  
XtSetArg(al[ac]. XmNx.300); ac--;  
XtSetArg(al[ac]. XmNy.100); ac--;  
XtSetValues(ab_help, al, ac);  
break;  
case 8 :  
ac=0;  
XtManageChild(mn_help);  
XtSetArg(al[ac]. XmNx.300); ac--;  
XtSetArg(al[ac]. XmNy.100); ac--;  
XtSetValues(mn_help, al, ac);  
break;  
case 9 :  
ac=0;  
XtManageChild(bb_browser);  
XtSetArg(al[ac]. XmNx.300); ac--;  
XtSetArg(al[ac]. XmNy.100); ac--;  
XtSetValues(bb_browser, al, ac);  
break;  
default :  
break;  
}
Widget menu:
{
  int ac;
  Arg a1[10];
  Widget item:

  ac = 0;
  XtSetArg(a1[ac], XmLabelString,
           XmStringCreateLtoR(item_name, char_set)); ac++;
  XtSetArg(a1[ac], XmNmnemonic.mnemonic); ac++;
  XtSetArg(a1[ac], XmNbackground, (XtArgVal) wheat.pixel); ac--;
     XtSetArg(a1[ac], XmNforeground, (XtArgVal) Blue.pixel); ac++;
  item = XmCreatePushButton(menu, item_name, a1, ac);
  XtManageChild(item);
  XtAddCallback(item, XmNactivateCallback, menuCB, client_data);
  XtSetSensitive(item, True);
  return(item);
}

="/******************************************************************/
Widget make_menu(menu_name, mnemonic, menu_bar)
  char *menu_name;
  char mnemonic;
  Widget menu_bar;
{
  int ac;
  Arg a1[10];
  Widget menu, cascade;

  menu= XmCreatePulldownMenu(menu_bar, menu_name, NULL, 0);
  ac=0;
  XtSetArg (a1[ac], XmNsubMenuId.menu, menu); ac++;
  XtSetArg (a1[ac], XmNmnemonic.mnemonic); ac++;
  XtSetArg (a1[ac], XmNLabelString,
            XmStringCreateLtoR(menu_name, char_set)); ac++;
  XtSetArg(a1[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac--;
     XtSetArg(a1[ac], XmNforeground, (XtArgVal) Blue.pixel); ac++;
  cascade=XmCreateCascadeButton(menu_bar, menu_name, a1, ac);
  XtManageChild(cascade);
  return(menu);
}

="/******************************************************************/
Widget make_help_menu (menu_name, mnemonic, menu_bar)
  char *menu_name;
  char mnemonic;
  Widget menu_bar;
{
  int ac;
  Arg a1[10];
  Widget menu, cascade;

  ac = 0;
  menu=XmCreatePulldownMenu(menu_bar, menu_name, a1, ac);
  ac=0;
  XtSetArg (a1[ac], XmNsubMenuId.menu); ac++;
  XtSetArg(a1[ac], XmNmemonic.mnemonic); ac++;
  XtSetArg(a1[ac], XmNLabelString,
           XmStringCreateLtoR(menu_name, XmSTRING_DEFAULT_CHARSET));
     ac++;

 125
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Blue.pixel); ac++;
cascade=XmCreateCascadeButton(menu_bar, menu_name, al, ac);
XtManageChild(cascade);

ac = 0;
XtSetArg (al[ac], XmNmenuHelpWidget, cascade); ac++;
XtSetValues (menu_bar, al, ac);

return(menu);
}  

/**

void create_menus(menu_bar)
Widget menu_bar;
{
    file_menu = make_menu("File", 'F', menu_bar);
    save_item = make_menu_item("Save", "Save", 'S', file_menu);
    empty_item = make_menu_item("Empty Document", "Empty", 'E', file_menu);
    quit_item = make_menu_item("Quit", "Quit", 'Q', file_menu);

    retrieve_menu = make_menu("Retrieve", 'R', menu_bar);
    retrieve_item =
        make_menu_item("Retrieve", "Retrieve", 'R', retrieve_menu);
    insert_menu = make_menu("Insert", 'I', menu_bar);
        insert_item = make_menu_item("Insert", "Insert", 'I', insert_menu);
    delete_menu = make_menu("Delete", 'D', menu_bar);
        delete_item = make_menu_item("Delete", "Delete", 'D', delete_menu);

    browse_menu = make_menu("Browse", 'B', menu_bar);
    browse_item = make_menu_item("Browse", "Browse", 'O', browse_menu);

    version_menu = make_menu("Version", 'V', menu_bar);
    version_item = make_menu_item("Version", "Version", 'E', version_menu);

    help_menu = make_help_menu("Help", 'H', menu_bar);
    about_item = make_menu_item("About", "About", 'A', help_menu);
    help_item = make_menu_item("Help", "Help", 'P', help_menu);

    add_accelerator(retrieve_item, "alt+r", "Alt<Key>r: 1");
    add_accelerator(insert_item, "alt+i", "Alt<Key>i: 1");
    add_accelerator(delete_item, "alt+d", "Alt<Key>d: 1");
    add_accelerator(quit_item, "alt+q", "Alt<Key>q: 1");
    add_accelerator(browse_item, "alt+b", "Alt<Key>b: 1");
    add_accelerator(version_item, "alt+v", "Alt<Key>v: 1");
    add_accelerator(about_item, "alt+a", "Alt<Key>a: 1");
    add_accelerator(help_item, "alt+p", "Alt<Key>p: 1");
    add_accelerator(save_item, "alt+s", "Alt<Key>s: 1");
    add_accelerator(empty_item, "alt+e", "Alt<Key>e: 1");

    XtSetSensitive(save_item, False);
    XtSetSensitive(empty_item, False);
}  

/**

void exact_dialog()
{

*/
Arg a1[20];
int ac;

ac = 0;
XtSetArg(a1[ac], XmNwidth, 400); ac++;
XtSetArg(a1[ac], XmNheight, 450); ac++;
bb_dialog2 = XmCreateBulletinBoardDialog(toplevel,
    "EXACT SOLUTIONS", a1, ac);

ac = 0;
XtSetArg(a1[ac], XmNListSizePolicy, XmCONSTANT); ac++;
XtSetArg(a1[ac], XmNselectionPolicy, XmSINGLE_SELECT); ac++;
XtSetArg(a1[ac], XmNtopItemPosition, 1); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) Wheat.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
list3 = XmCreateScrolledList(bb_dialog2, "list3", a1, ac);
XtManageChild(list3);
XtAddCallback(list3, XmNSingleSelectionCallback, solutionCB, EXACT);

ac = 0;
XtSetArg(a1[ac], XmNheight, 300); ac++;
XtSetArg(a1[ac], XmNwidth, 375); ac++;
XtSetValues(list3, a1, ac);

ac = 0;
XtSetArg(a1[ac], XmNpacking, XmPACK_COLUMN); ac++;
XtSetArg(a1[ac], XmNorientation, XmHORIZONTAL); ac++;
XtSetArg(a1[ac], XmNnumColumns, 1); ac++;
XtSetArg(a1[ac], XmNadjustLast, False); ac++;
XtSetArg(a1[ac], XmNentryAlignment, XmALIGNMENT_CENTER); ac++;
XtSetArg(a1[ac], XmNspacing, 50); ac++;
rowcoll = XmCreateRowColumn(bb_dialog2, "rowcoll", a1, ac);
XtManageChild(rowcoll);

ac = 0;
XtSetArg(a1[ac], XmNLabelString,
    XmStringCreate("General", char_set)); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
butt[0] = XmCreatePushButton(rowcoll, "butt0", a1, ac);
XtManageChild(butt[0]);
XtAddCallback(butt[0], XmNActivateCallback, dialog2CB, GENERAL);

ac = 0;
XtSetArg(a1[ac], XmNLabelString,
    XmStringCreate("Specific", char_set)); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
butt[1] = XmCreatePushButton(rowcoll, "butt1", a1, ac);
XtManageChild(butt[1]);
XtAddCallback(butt[1], XmNActivateCallback, dialog2CB, SPECIFIC);

ac = 0;
XtSetArg(a1[ac], XmNLabelString,
    XmStringCreate("Quit", char_set)); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
butt[2] = XmCreatePushButton(rowcoll, "butt2", a1, ac);
XtManageChild(butt[2]);
XtAddCallback(butt[2], XmNactivateCallback, diaoq2CB, QUIT);

ac = 0;
XtSetArg(a1[ac], XmNX, 1); ac++;
XtSetArg(a1[ac], XmNY, 1); ac++;
XtSetValues(list3, a1, ac);

ac = 0;
XtSetArg(a1[ac], XmNX, 50); ac++;
XtSetArg(a1[ac], XmNY, 400); ac++;
XtSetValues(rowcoll, a1, ac);
}
/**********************************************************************************/
void no_general_solutions()
{
Arg a1[20];
int ac;

ac = 0;
XtSetArg(a1[ac], XmNmessageString,
XmStringCreateLtoR("No General Function(s) Found", char_set)); ac++;
XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
XtSetArg(a1[ac], XmNbackgroundColor, (XtArgVal) Red.pixel); ac++;
XtSetArg(a1[ac], XmNforeground, (XtArgVal) Yellow.pixel); ac++;
dia2 = XmCreateErrorDialog(toplevel, "SOLUTION(S)", a1, ac);
XtUnmanageChild(XmMessageBoxGetChild(dia2, XmDIALOOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxGetChild(dia2, XmDIALOOGANCEL_BUTTON));
}
/**********************************************************************************/
void no_specific_solutions()
{
Arg a1[20];
int ac;

ac = 0;
XtSetArg(a1[ac], XmNmessageString,
XmStringCreateLtoR("No Specific Function(s) Found", char_set)); ac++;
XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
XtSetArg(a1[ac], XmNbackgroundColor, (XtArgVal) Red.pixel); ac++;
XtSetArg(a1[ac], XmNforeground, (XtArgVal) Yellow.pixel); ac++;
dia2 = XmCreateErrorDialog(toplevel, "SOLUTION(S)", a1, ac);
XtUnmanageChild(XmMessageBoxGetChild(dia2, XmDIALOOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxGetChild(dia2, XmDIALOOGCANCEL_BUTTON));
}
/**********************************************************************************/
void no_freeze_solutions()
{
Arg a1[20];
int ac;

ac = 0;
XtSetArg(a1[ac], XmNmessageString,
XmStringCreateLtoR("No Function(s) Found By freezing Argument(s)",
char_set));
ac++;
XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
XtSetArg(a1[ac], XmNbackgroundColor, (XtArgVal) Red.pixel); ac++;
}
Program Listing

XtSetArg(al[ac], XmNforeground, (XtArgVal) Yellow.pixel); ac++;
dial2 = XmCreateErrorsDialog(toplevel, "SOLUTION(S)", al, ac);
XtUnmanageChild(XMessageBoxGetChild(dial2, XmDIALOG_ERROR_BUTTON));
XtUnmanageChild(XMessageBoxGetChild(dial2, XmDIALOG_CANCEL_BUTTON));

/***********************************************************************/
void no_delete_match()
{
Arg al[20];
int ac;

ac = 0;
XtSetArg(al[ac], XmNmessageString,
 XmStringCreateLtoR("No such type exists", char_set)); ac++;
XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) Red.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Yellow.pixel); ac++;
no_del = XmCreateErrorDialog(toplevel, "NO_MATCH", al, ac);
XtUnmanageChild(XMessageBoxGetChild(no_del, XmDIALOG_ERROR_BUTTON));
XtUnmanageChild(XMessageBoxGetChild(no_del,
    XmDIALOG_CANCEL_BUTTON));

/***********************************************************************/
void insert_match()
{
Arg al[20];
int ac;

ac = 0;
XtSetArg(al[ac], XmNmessageString,
 XmStringCreateLtoR("Function with similar type and name
 already exists", char_set)); ac++;
XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) Red.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Yellow.pixel); ac++;
in_s MATCH = XmCreateErrorDialog(toplevel, "INSERTION_MATCH", al, ac);
XtUnmanageChild(XMessageBoxGetChild(in_s MATCH, XmDIALOG_ERROR_BUTTON));
XtUnmanageChild(XMessageBoxGetChild(in_s MATCH,
    XmDIALOG_CANCEL_BUTTON));

/***********************************************************************/
void no_name()
{
Arg al[20];
int ac;

ac = 0;
XtSetArg(al[ac], XmNmessageString,
 XmStringCreateLtoR("No such name corresponds to given type", char_set)); ac++;
XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
in no name del = XmCreateErrorDialog(toplevel, "NO_NAME", al, ac);
XtUnmanageChild(XMessageBoxGetChild(no name del,
    XmDIALOG_ERROR_BUTTON));
XtUnmanageChild(XMessageBoxGetChild(no name del,
    XmDIALOG_CANCEL_BUTTON));

/***********************************************************************/
void version()
Program Listing

```c
[ Arg al[20];
int ac;

ac = 0;
XtSetArg(al[ac], XmNmessageString,
XmStringCreateLtoR("Version : 1.0\nProgrammer
: Deoraq Ramjisingh\nDeveloped At
University Of
Windsor\nWindsor, ON. Canada N9B
3P4\nInterface language : Motif\nImplementation language : C", char_set)); ac++;
XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_BEGINNING); ac++;
ver = XmCreateMessageDialog(toplevel, "VERSION", al, ac);
XtUnmanageChild(XmMessageBoxGetChild(ver, XmDIALOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxGetChild(ver, XmDIALOG_CANCEL_BUTTON));
}
/**************************************************************************/
void about_help()
{
  Arg al[20];
  int ac:

  ac = 0;
  XtSetArg(al[ac], XmNmessageString,
  XmStringCreateLtoR("Windsor Software for Reuse (WISER) is a
  software retrieval system that is used in the construction of
  Miranda programs. by combining functions(s). This system allows
  the user to formulate queries, based on types as search keys,
  so as to obtain relevant functions, that could be of importance
  to the program being constructed. Added to the retrieval
  option, a number of other useful options are also available to the
  user, such as: insertion of a function, deletion of a function
  and the deletion of a type along with it's associated functions.
  This program deals with persistent data, hence, the database
  consists of number of files that are used for the storage of
  our records. A record consists of a number of components as shown
  in the example below. Furthermore, the database is continuously
  evolving, by making use of user's queries. Example of a record:
  ([a] -> a) [hd, last, limit, max, min] [250] [5250, 8250, 10000] [5750]
  The interface is designed with the hope of being user friendly, and
  as such, provides a number of help buttons where need be.",
  char_set)); ac++;
  XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_BEGINNING); ac++;
  ab_help = XmCreateMessageDialog(toplevel, "ABOUT HELP", al, ac);
  XtUnmanageChild(XmMessageBoxGetChild(ab_help, XmDIALOG_HELP_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(ab_help,
  XmDIALOG_CANCEL_BUTTON));
}
/**************************************************************************/
void main_help()
{
 Arg al[20];
 int ac:

 ac = 0;
 XtSetArg(al[ac], XmNmessageString,
 XmStringCreateLtoR("Make your selection from the menu bar
 by using either the keyboard (by meta_key) or the mouse.\n
 130
```
Sufficient documentation is available for the retrieval, insertion, and deletion processes. But just as a reminder, during the retrieval process a list of function's type(s) and name(s) might appear, if they exist. The user can see the documentation of the required functions by clicking the LEFT mouse button once on their position(s) in the list. The browser for the system has been implemented.

```c
XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_BEGINNING); ac++;
```

```c
mn_help = XmCreateMessageDialog(toplevel, "HELP", a1, ac);
XtUnmanageChild(XmMessageBoxBoxGetChild(mn_help, XmDIALOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxBoxGetChild(mn_help, XmDIALOG_CANCEL_BUTTON));
```

}/*.........................................................................*/

```c
void help_retrieve()
{
Arq a1[20];
int ac;
ac = 0;
XtSetArg(a1[ac], XmNmessageString, XmStringCreateLtoR("(1). Slide the horizontal scroll bar labelled. 'Number Of Arguments', to correspond to the number of arguments in the function desired. Doing so will allow for a list of arguments to be displayed.(2). Move the mouse so that the cursor is positioned over the first argument in the list. Then click on the LEFT mouse button, this will cause the current content of that argument to be displayed in the box labelled 'Argument Type:'.(3). Move the cursor to the box labelled 'Argument Type:', and click on the LEFT mouse button.(4). The content of that box, which corresponds to a particular argument, can now be edited.(5). When finish editing or entering that argument's type, press the enter key on the keyboard.(6). Repeat (2). to (5). for other arguments.(7). If an incorrect type is detected in the list after editing, then correction can be made by following (2). to (5). for that particular argument.(8). Move the cursor to the box labelled 'Result Type:', and click the LEFT mouse button, to allow for editing.(9). Edit or enter the type of the result, for the function desired.(10). There is no need to press the return key in this box, since it is inoperative here.(11). If everything seems correct then press the 'APPLY' button with the LEFT mouse button.(12). To reset the retrieval window to its original state then press the 'RESET' button with the LEFT mouse button.(13). To terminate the retrieval process press the 'QUIT' button with the LEFT mouse button."), char_set)); ac++;
```

```c
XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_BEGINNING); ac++;
help_ret = XmCreateMessageDialog(toplevel, "RETRIEVE HELP", a1, ac);
XtUnmanageChild(XmMessageBoxBoxGetChild(help_ret, XmDIALOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxBoxGetChild(help_ret, XmDIALOG_CANCEL_BUTTON));
```

}/*.........................................................................*/

```c
void help_insert()
{
Arq a1[20];
int ac;
```
ac = 0;
XtSetArg(al[ac], XmNmessageString,
XmStringCreateLtoR("(1). Slide the horizontal scroll bar labelled, 
'Number Of Arguments', to correspond to the
number of arguments in the function being inserted. Doing so will allow for 
a list of arguments to be displayed.\n(2). Move the mouse so that, the cursor is positioned over the first argument in the 
list.\nThen click on the LEFT mouse button, this will cause 
the current content of that argument to be displayed in the 
box labelled 'Argument Type':.\n(3). Move the cursor to the box labelled, 'Argument Type:', and click on the LEFT mouse\nbutton.\nThe content of that box, which corresponds to a 
particular argument, can now be edited.\n(5). When finish 
editing or entering that argument's type, press the enter key on 
the keyboard.\n(6). Repeat (2), to (5), for other arguments. 
\n(7). If an incorrect type is detected in the list after editing, 
then correction can be made by following (2), to (5), for that 
particular argument.\n(8). Move the cursor to the box labelled, 'Result Type:', and click the LEFT mouse button,\nto allow for 
editing.\n(9). Edit or enter the type of the result, for the 
function being inserted.\n(10). There is no need to press the 
return key in this box, since it is inoperative here.\n(11). Move 
the cursor to the box labelled 'Function Name:', and click the 
LEFT mouse button,\nto allow for editing.\n(12). Edit or 
enter the name of the function being inserted.\n(13). see 
(10)\n(14). If everything seems correct then press the 'APPLY' 
button with the LEFT mouse button.\n(15). To reset the retrieval window to its original state then press the 'RESET' button\nwith the LEFT mouse button.\n(16). To terminate the insertion 
process press the 'QUIT' button with the LEFT mouse button."
,
char_set); ac++;
XtSetArg(al[ac], XmNmessageAlignment, XmALIGNMENT_BEGINNING); ac++;
help_ins = XmCreateMessageDialog(toplevel, "INSERT HELP", al, ac);
XtUnmanageChild(XmMessageBoxGetChild(help_ins,
XMDIALOG_HELP_BUTTON));
XtUnmanageChild(XmMessageBoxGetChild(help_ins,
XMDIALOG_CANCEL_BUTTON));
}

**************************************************************************

void help_delete()
{

Arq al[20];
int ac;

ac = 0;
XtSetArg(al[ac], XmNmessageString,
XmStringCreateLtoR("(1). Slide the horizontal scroll bar labelled, 
'Number Of Arguments', to correspond to the
number of arguments in the function being deleted. Doing so will allow for 
a list of arguments to be displayed.\n(2). Move the mouse so that, the cursor is positioned over the first argument in the 
list.\nThen click on the LEFT mouse button, this will cause 
the current content of that argument to be displayed in the 
box labelled 'Argument Type':.\n(3). Move the cursor to the box labelled, 'Argument Type:', and click on the LEFT mouse\nbutton.\nThe content of that box, which corresponds to a 
particular argument, can now be edited.\n(5). When finish 
editing or entering that argument's type, press the enter key on 
the keyboard.\n(6). Repeat (2), to (5), for other arguments.\n
(7). If an incorrect type is detected in the list after editing, then correction can be made by following (2), (5), and (9). For that particular argument. Move the cursor to the box labelled, ‘Result Type:’, and click the LEFT mouse button to allow for editing. There is no need to press the return key in this box, since it is inoperative here. Move the cursor to the box labelled ‘Function Name:’, and click the mouse button to allow for editing. If everything seems correct then press the ‘APPLY’ button with the LEFT mouse button. To reset the retrieval window to its original state then press the ‘RESET’ button with the LEFT mouse button. To terminate the deletion process press the ‘QUIT’ button with the LEFT mouse button.

} /*----------------------------------------------------------*/

void retrieve_dialog()
{
    Arg a[20];
    int ac;
    XFontStruct *font=NULL;
    XmFontList fontlist =NULL;
    char *namestring =NULL;

    ac=0;
    XtSetArg(a[ac], XmNwidth, 305); ac++;
    XtSetArg(a[ac], XmNheight, 368); ac++;
    bb_dialog = XmCreateBulletinBoardDialog(toplevel, "RETRIEVE", a, ac);

    ac = 0;
    XtSetArg(a[ac], XmNWidth, 280); ac++;
    sep4 = XmCreateSeparator(bb_dialog, "sep4", a, ac);
    XtManageChild(sep4);

    ac = 0;
    XtSetArg(a[ac], XmNWidth, 280); ac++;
    sep1 = XmCreateSeparator(bb_dialog, "sep1", a, ac);
    XtManageChild(sep1);

    ac = 0;
    XtSetArg(a[ac], XmNWidth, 280); ac++;
    sep2 = XmCreateSeparator(bb_dialog, "sep2", a, ac);
    XtManageChild(sep2);

    ac = 0;
    XtSetArg(a[ac], XmNWidth, 280); ac++;
    sep3 = XmCreateSeparator(bb_dialog, "sep3", a, ac);
    XtManageChild(sep3);

    ac = 0;
Program Listing

XtSetArg(al[ac], XmNtitleString, 
XmStringCreate("Number Of Arguments", char_set)); ac++; 
XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++; 
XtSetArg(al[ac], XmNshowValue, True); ac++; 
XtSetArg(al[ac], XmNminimum, 0); ac++; 
XtSetArg(al[ac], XmNmaximum, 8); ac++; 
XtSetArg(al[ac], XmNheight, 60); ac++; 
XtSetArg(al[ac], XmNwidth, 275); ac++; 
scale = XmCreateScale(bb_dialog, "scale", al, ac); 
 XtManageChild(scale); 
 XtAddCallback(scale, XmNvalueChangedCallback, handle_scale, NULL); 

frame1 = XmCreateFrame(bb_dialog, "frame1", NULL, 0); 
XtManageChild(frame1); 

frame2 = XmCreateFrame(bb_dialog, "frame2", NULL, 0); 
XtManageChild(frame2); 

ac = 0; 
XtSetArg(al[ac], XmNlistSizePolicy, XmCONSTANT); ac++; 
XtSetArg(al[ac], XmNselectionPolicy, XmSINGLE_SELECT); ac++; 
XtSetArg(al[ac], XmNtopItemPosition, 1); ac++; 
list = XmCreateScrolledList(frame1, "list", al, ac); 
XtManageChild(list); 
XtAddCallback(list, XmNsingleSelectionCallback, selectCB, NULL); 

ac = 0; 
XtSetArg(al[ac], XmNheight, 90); ac++; 
XtSetArg(al[ac], XmNwidth, 268); ac++; 
XtSetValues(list, al, ac); 
frame = XmCreateFrame(bb_dialog, "frame", NULL, 0); 
XtManageChild(frame); 

ac = 0; 
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++; 
XtSetArg(al[ac], XmNheight, 35); ac++; 
XtSetArg(al[ac], XmNwidth, 190); ac++; 
text = XmCreateText(frame, "text", al, ac); 
XtManageChild(text); 
XtAddSensitive(text, False); 
XtAddCallback(text, XmNactivateCallback, TextCB, NULL); 

XtSetArg(al[ac], XmNlabelString, 
XmStringCreateLtoR("Argument Type:", char_set)); ac++; 
XtSetArg(al[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++; 
label2 = XmCreateLabel(bb_dialog, "label2", al, ac); 
XtManageChild(label2); 

XtSetArg(al[ac], XmNlabelString, 
XmStringCreateLtoR("Result Type:", char_set)); ac++; 
XtSetArg(al[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++; 
label1 = XmCreateLabel(bb_dialog, "label1", al, ac); 
XtManageChild(label1); 

ac = 0; 
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++; 
XtSetArg(al[ac], XmNheight, 35); ac++; 
XtSetArg(al[ac], XmNwidth, 190); ac++; 
text1 = XmCreateText(frame2, "text1", al, ac);
Program Listing

XtManageChild(text1);

namestring = "*times*18*";
font = XLoadQueryFont(XtDisplay(label12), namestring);
fontlist = XmFontListCreate(font, char_set);

ac = 0;
XtSetArg(al[ac], XmNpacking, XmPACK_COLUMN); ac++;
XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++;
XtSetArg(al[ac], XmNnumColumns, 1); ac++;
XtSetArg(al[ac], XmNadjustLast, False); ac++;
rowcol = XmCreateRowColumn(bb_dialog, "rowcol", al, ac);
XtManageChild(rowcol);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Apply", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
bbuttons[0] = XmCreatePushButton(rowcol, "button0", al, ac);
XtManageChild(bbuttons[0]);
XtAddCallback(bbuttons[0], XmNactivateCallback, buttonCB, APPLY);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Reset", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
bbuttons[1] = XmCreatePushButton(rowcol, "button1", al, ac);
XtManageChild(bbuttons[1]);
XtAddCallback(bbuttons[1], XmNactivateCallback, buttonCB, RESET);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Quit", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
bbuttons[2] = XmCreatePushButton(rowcol, "button2", al, ac);
XtManageChild(bbuttons[2]);
XtAddCallback(bbuttons[2], XmNactivateCallback, buttonCB, QUIT);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreate("Help...", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
bbuttons[3] = XmCreatePushButton(rowcol, "button3", al, ac);
XtManageChild(bbuttons[3]);
XtAddCallback(bbuttons[3], XmNactivateCallback, buttonCB, HELP);

/**---------------------------------------------*/
void insert_dialog()
{
    Arg al[20];
Program Listing

int ac;
XFontStruct *font=NULL;
XmFontList fontlist =NULL;
char *namestring=NULL;
ac=0;
XtSetArg(a1[ac],XmNwidth,305); ac--;
XtSetArg(a1[ac],XmNheight,423); ac--;
   bbDia = XmCreateBulletinBoardDialogiallylevel,"INSERT",a1,ac); 
ac = 0;
XtSetArg(a1[ac],XmNWidth,280); ac--;
s4 = XmCreateSeparator(bbDia,"s4",a1,ac);
    XtManageChild(s4);
ac = 0;
XtSetArg(a1[ac],XmNWidth,280); ac--;
s1 = XmCreateSeparator(bbDia,"s1",a1,ac);
    XtManageChild(s1);
ac = 0;
XtSetArg(a1[ac],XmNWidth,280); ac--;
s2 = XmCreateSeparator(bbDia,"s2",a1,ac);
    XtManageChild(s2);
ac = 0;
XtSetArg(a1[ac],XmNWidth,280); ac--;
s3 = XmCreateSeparator(bbDia,"s3",a1,ac);
    XtManageChild(s3);
ac =0;
XtSetArg(a1[ac],XmNtitleString,
    XmStringCreate("Number Of Arguments",x,ac_set)); ac++;
XtSetArg(a1[ac],XmNoRientation,XmHORIZONTAL); ac++;
XtSetArg(a1[ac],XmNshowValue,True); ac++;
XtSetArg(a1[ac],XmNminimum,0); ac++;
XtSetArg(a1[ac],XmNmaximum,8); ac++;
XtSetArg(a1[ac],XmNheight,60); ac++;
XtSetArg(a1[ac],XmNwidth,275); ac++;
scal=XmCreateScale(bbDia,"scal",a1,ac);
    XtManageChild(scal);
    XTAddCallback(scal,XmNvalueChangedCallback,
        handle_scale_insert,NULL);
   f1 = XmCreateFrame(bbDia,"f1",NULL,0);
    XtManageChild(f1);
   f2 = XmCreateFrame(bbDia,"f2",NULL,0);
    XtManageChild(f2);
   f4 = XmCreateFrame(bbDia,"f4",NULL,0);
    XtManageChild(f4);
ac = 0;
XtSetArg(a1[ac],XmNListSizePolicy,XmCONSTANT); ac++;
XtSetArg(a1[ac],XmNselectionPolicy,XmSINGLE_SELECT); ac++;
XtSetArg(a1[ac],XmNtopItemPosition,1);ac++;
lis = XmCreateScrolledList(f1,"lis",a1,ac);
Program Listing

XtManageChild(list);
XtAddCallback(list, XmSelectionCallback, selectCB_insert, NULL);

ac = 0;
XtSetArg(al[ac], XmNheight, 90); ac++;
XtSetArg(al[ac], XmNwidth, 268); ac++;
XtSetValues(list.al[ac],
    f = XmCreateFrame(bb_dia, "f", NULL, 0);
XtManageChild(f);

ac = 0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
tex = XmCreateText(f, "text", al.ac);
XtManageChild(tex);
XtSetSensitive(tex, false);
XtAddCallback(tex, XmNactivateCallback, TextCB_insert, NULL);

XtSetArg(al[ac], XmNlabelString,
    XmStringCreateLtoR("Argument Type:" , char_set)); ac++;
XtSetArg(al[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
12 = XmCreateLabel(bb_dia, "12", al.ac);
XtManageChild(12);

XtSetArg(al[ac], XmNlabelString,
    XmStringCreateLtoR("Result Type:" , char_set)); ac++;
XtSetArg(al[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
11 = XmCreateLabel(bb_dia, "11", al.ac);
XtManageChild(11);

ac = 0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
tex1 = XmCreateText(f2, "tex1", al.ac);
XtManageChild(tex1);

XtSetArg(al[ac], XmNlabelString,
    XmStringCreateLtoR("Function Name:" , char_set)); ac++;
XtSetArg(al[ac], XmNalignment, XmALIGNMENT_BEGINNING); ac++;
14 = XmCreateLabel(bb_dia, "14", al.ac);
XtManageChild(14);

ac = 0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
tex4 = XmCreateText(f4, "text4", al.ac);
XtManageChild(tex4);

namestring = "times*18";
font = XLoadQueryFont(XDisplay(12), namestring);
fontlist = XmFontListCreate(font, char_set);

ac = 0;
XtSetArg(al[ac], XmNpacking, XmPACK_COLUMN); ac++;
XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++;
XtSetArg(al[ac], XmNnumColumns, 1); ac++;

137
void delete_dialog()
{
    Arg a1[20];
    int ac;
    XFontStruct *font=NULL;
    XmFontList fontlist =NULL;
    char *namestring =NULL;

    ac=0;
    XtSetArg(a1[ac], XmNwidth, 305); ac++;
    XtSetArg(a1[ac], XmNheight, 423); ac++;
    bb_dialog_del =
        XmCreateBulletinBoardDialog(toplevel, "DELETE", a1, ac);
}
ac = 0;
XtSetArg(al[ac], XmNwidth, 280); ac++;
se4 = XmCreateSeparator(bb_dia, "se4", al, ac);
XtManageChild(se4);

ac = 0;
XtSetArg(al[ac], XmNwidth, 280); ac++;
se1 = XmCreateSeparator(bb_dialog, "se1", al, ac);
XtManageChild(se1);

ac = 0;
XtSetArg(al[ac], XmNwidth, 280); ac++;
se2 = XmCreateSeparator(bb_dialog, "se2", al, ac);
XtManageChild(se2);

ac = 0;
XtSetArg(al[ac], XmNwidth, 280); ac++;
se3 = XmCreateSeparator(bb_dialog, "se3", al, ac);
XtManageChild(se3);

ac = 0;
XtSetArg(al[ac], XmNtitleString, XmNlabelString, "Number Of Arguments", char_set); ac++;
XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++;
XtSetArg(al[ac], XmNshowValue, True); ac++;
XtSetArg(al[ac], XmNminimum, 0L); ac++;
XtSetArg(al[ac], XmNmaximum, 8L); ac++;
XtSetArg(al[ac], XmNheight, 60); ac++;
XtSetArg(al[ac], XmNwidth, 275); ac++;
sca = XmCreateScale(bb_dialog, "sca", al, ac);
XtManageChild(sca);
XtAddCallback(sca, XmNValueChangedCallback, handle_scale_delete, NULL);

fr1 = XmCreateFrame(bb_dialog, "fr1", NULL, 0);
XtManageChild(fr1);

fr2 = XmCreateFrame(bb_dialog, "fr2", NULL, 0);
XtManageChild(fr2);

fr4 = XmCreateFrame(bb_dialog, "fr4", NULL, 0);
XtManageChild(fr4);

ac = 0;
XtSetArg(al[ac], XmNListSizePolicy, XmCONSTANT); ac++;
XtSetArg(al[ac], XmNSelectionPolicy, XmSINGLE_SELECT); ac++;
XtSetArg(al[ac], XmNtopItemPosition, 1); ac++;
l1 = XmCreateScrolledList(fr1, "l1", al, ac);
XtManageChild(l1);
XtAddCallback(l1, XmNSelectionCallback, selectCB_delete, NULL);

ac = 0;
XtSetArg(al[ac], XmNheight, 90); ac++;
XtSetArg(al[ac], XmNwidth, 268); ac++;
XtSetValues(l1, al, ac);
fr = XmCreateFrame(bb_dialog, "fr", NULL, 0);
XtManageChild(fr);

ac = 0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;

139
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
t= XmCreateText(fr, "te", al, ac);
 XtManageChild(t);
XtSetSensitive(t, False);
XtAddCallback(t, XmNactivateCallback, TextCB_delete, NULL);

XtSetArg(al[ac], XmNLabelString,
         XmStringCreateToString("Argument Type: ", char_set)); ac++;
XtSetArg(al[ac], XmNLabelAlignment, XmALIGMENT_BEGINNING); ac++;
l1=xmCreateLabel(bb_dialog_del, "la2", al, ac);
 XtManageChild(l1);

XtSetArg(al[ac], XmNLabelString,
         XmStringCreateToString("Result Type: ", char_set)); ac++;
XtSetArg(al[ac], XmNLabelAlignment, XmALIGNMENT_BEGINNING); ac++;
l1=xmCreateLabel(bb_dialog_del, "la1", al, ac);
 XtManageChild(l1);

ac =0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
tел=xmCreateText(fr2, "tel", al, ac);
 XtManageChild(tел);

XtSetArg(al[ac], XmNLabelString,
         XmStringCreateToString("Function Name: ", char_set)); ac++;
XtSetArg(al[ac], XmNLabelAlignment, XmALIGNMENT_BEGINNING); ac++;
l1=xmCreateLabel(bb_dialog_del, "la4", al, ac);
 XtManageChild(la4);

ac =0;
XtSetArg(al[ac], XmNeditMode, XmSINGLE_LINE_EDIT); ac++;
XtSetArg(al[ac], XmNheight, 35); ac++;
XtSetArg(al[ac], XmNwidth, 190); ac++;
tex4=xmCreateText(fr4, "tex4", al, ac);
 XtManageChild(tex4);

namestring = "*times*18*";
font = XLoadQueryFont(XtDisplay(la2), namestring);
fontlist=XmFontListCreate(font, char_set);

ac =0;
XtSetArg(al[ac], XmNpacking, XmPACK_COLUMN); ac++;
XtSetArg(al[ac], XmNorientation, XmHORIZONTAL); ac++;
XtSetArg(al[ac], XmNumColumns, 1); ac++;
XtSetArg(al[ac], XmNadjJustLast, False); ac++;
rc = XmCreateRowColumn(bb_dialog_del, "rc", al, ac);
 XtManageChild(rc);

ac = 0;
XtSetArg(al[ac], XmNLabelString,
         XmStringCreate("Apply", char_set)); ac++;
XtSetArg(al[ac], XmNFontList, fontlist); ac--;
XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac--;
XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac--;
but[0] = XmCreatePushButton(rc, "bu0", al, ac);
XtManageChild(bu[0]);
XtAddCallback(bu[0], XmNactivateCallback.buttonCB_delete, APPLY);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
    XmStringCreate("Reset", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
    bu[1] = XmCreatePushButton(rc, "bu1", al, ac);
XtManageChild(bu[1]);
XtAddCallback(bu[1], XmNactivateCallback.buttonCB_delete, RESET);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
    XmStringCreate("Quit", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
    bu[2] = XmCreatePushButton(rc, "bu2", al, ac);
XtManageChild(bu[2]);
XtAddCallback(bu[2], XmNactivateCallback.buttonCB_delete, QUIT);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
    XmStringCreate("Help...", char_set)); ac++;
XtSetArg(al[ac], XmNfontList, fontlist); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
    bu[3] = XmCreatePushButton(rc, "b3", al, ac);
XtManageChild(bu[3]);
XtAddCallback(bu[3], XmNactivateCallback.buttonCB_delete, HELP);

void general_sol_dialog()
{
    Arg al[20];
    int ac;

    ac = 0;
    XtSetArg(al[ac], XmNWidth, 400); ac++;
    XtSetArg(al[ac], XmNHeight, 450); ac++;
    bb_dialog1 = XmCreateBulletinBoardDialog(toplevel,
        "GENERAL SOLUTION(S)", al, ac);

ac = 0;
    XtSetArg(al[ac], XmNListSizePolicy, XmCONSTANT); ac++;
    XtSetArg(al[ac], XmNselectionPolicy, XmSINGLE_SELECT); ac++;
    XtSetArg(al[ac], XmNtopItemPosition, 1); ac++;
    XtSetArg(al[ac], XmNbackground, (XtArgVal) Wheat.pixel); ac++;
    XtSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac++;
    list1 = XmCreateScrolledList(bb_dialog1, "list1", al, ac);
    XtManageChild(list1);
    XtAddCallback(list1, XmNsingleSelectionCallback, solutionCB, GENERAL);

ac = 0;
    XtSetArg(al[ac], XmNHeight, 300); ac++;
}
XTSetArg(al[ac], XmNwidth, 375); ac--;
XTSetValues(list1, al, ac);

ac = 0;
XTSetArg(al[ac], XmNLabelString,
XmStringCreate("Quit", char_set)); ac--;
XTSetArg(al[ac], XmNbackgroundColor, (XtArgVal) LimeGreen.pixel); ac--;
XTSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac--;
butto = XmCreatePushButton(bb_dialog1, "butto", al, ac);
XTManageChild(butto);
XTAddCallback(butto, XmNActivateCallback, dialog3CB, GENERAL);

ac = 0;
XTSetArg(al[ac], XmNx, 1); ac--;
XTSetArg(al[ac], XmNy, 1); ac--;
XTSetValues(list1, al, ac);

ac = 0;
XTSetArg(al[ac], XmNx, 180); ac--;
XTSetArg(al[ac], XmNy, 400); ac--;
XTSetValues(butto, al, ac);
}

void specific_sol_dialog()
{
Arg al[20];
int ac;

ac = 0;
XTSetArg(al[ac], XmNwidth, 400); ac++;
XTSetArg(al[ac], XmNheight, 450); ac++;
bb_dial = XmCreateBulletinBoardDialog(top_level,
"SPECIFIC SOLUTION(S)", al, ac);

ac = 0;
XTSetArg(al[ac], XmNlistSizePolicy, XmCONSTANT); ac--;
XTSetArg(al[ac], XmNselectionPolicy, XmSINGLE_SELECT); ac--;
XTSetArg(al[ac], XmNtopItemPosition, 1); ac--;
XTSetArg(al[ac], XmNbackgroundColor, (XtArgVal) Wheat.pixel); ac--;
XTSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac--;
list1 = XmCreateScrolledList(bb_dial, "list1", al, ac);
XTManageChild(list1);
XTAddCallback(list1, XmNsingleSelectionCallback, solutionCB, SPECIFIC);

ac = 0;
XTSetArg(al[ac], XmNheight, 300); ac--;
XTSetArg(al[ac], XmNwidth, 375); ac--;
XTSetValues(list1, al, ac);

ac = 0;
XTSetArg(al[ac], XmNLabelString,
XmStringCreate("Quit", char_set)); ac--;
XTSetArg(al[ac], XmNbackgroundColor, (XtArgVal) LimeGreen.pixel); ac--;
XTSetArg(al[ac], XmNforeground, (XtArgVal) Black.pixel); ac--;
buttl = XmCreatePushButton(bb_dial, "buttl", al, ac);
XTManageChild(buttl);
XTAddCallback(buttl, XmNActivateCallback, dialog3CB, SPECIFIC);
ac = 0;
XtSetArg(a1[ac], XmNValue, 1); ac++;
XtSetArg(a1[ac], XmNValue, 1); ac++;
XtSetValues(list1, a1, ac);

ac = 0;
XtSetArg(a1[ac], XmNValue, 180); ac++;
XtSetArg(a1[ac], XmNValue, 400); ac++;
XtSetValues(button1, a1, ac);
}
/* *****************************************************************************/

void freeze_solution()
{
    Arg a1[20];
    int ac;

    ac=0;
    XtSetArg(a1[ac], XmNWidth, 400); ac++;
    XtSetArg(a1[ac], XmNHeight, 450); ac++;
    bb_dial = XmCreateBulletinBoardDialog(top_level,
        "FREEZING ARG(S) SOLUTION(S)", a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNListSizePolicy, XmCONSTANT); ac++;
    XtSetArg(a1[ac], XmNSelectionPolicy, XmSINGLE_SELECT); ac++;
    XtSetArg(a1[ac], XmNTopItemPosition, 1); ac++;
    XtSetArg(a1[ac], XmNBackground, (XtArgVal) Wheat.pixel); ac++;
    XtSetArg(a1[ac], XmNForeground, (XtArgVal) Black.pixel); ac++;
    ill = XmCreateScrolledList(bb_dial, "ill", a1, ac);
    XtManageChild(ill);
    XtAddCallback(ill, XmNsingleSelectionCallback, solutionCB, FREEZE);

    ac = 0;
    XtSetArg(a1[ac], XmNHeight, 300); ac++;
    XtSetArg(a1[ac], XmNWidth, 375); ac++;
    XtSetValues(ill, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNLabelString,
        XmStringCreate("Quit", char_set)); ac++;
    XtSetArg(a1[ac], XmNBackground, (XtArgVal) LimeGreen.pixel); ac++;
    XtSetArg(a1[ac], XmNForeground, (XtArgVal) Black.pixel); ac++;
    butt2 = XmCreatePushButton(bb_dial, "butt2", a1, ac);
    XtManageChild(butt2);
    XtAddCallback(butt2, XmNactivateCallback, dialog3CB, FREEZE);

    ac = 0;
    XtSetArg(a1[ac], XmNValue, 1); ac++;
    XtSetArg(a1[ac], XmNValue, 1); ac++;
    XtSetValues(list1, a1, ac);

    ac = 0;
    XtSetArg(a1[ac], XmNValue, 180); ac++;
    XtSetArg(a1[ac], XmNValue, 400); ac++;
    XtSetValues(button2, a1, ac);
}
/* *****************************************************************************/
void no_exact_dialog()
{
    Arg a1[20];
    int ac;

    ac = 0;
    XtSetArg(a1[ac], XmNmessageString,
             XmStringCreateLtoR("No exact match found", char_set)); ac++;
    XtSetArg(a1[ac], XmNcancelLabelString,
             XmStringCreateLtoR("Quit", char_set)); ac++;
    XtSetArg(a1[ac], XmNokLabelString,
             XmStringCreateLtoR("General", char_set)); ac++;
    XtSetArg(a1[ac], XmNhelpLabelString,
             XmStringCreateLtoR("Specific", char_set)); ac++;
    XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) Orange.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Navy.pixel); ac++;
    dialog = XmCreateMessageDialog(toplevel, "SOLUTION", a1, ac);
    XtAddCallback(dialog, XmNokCallback, dialogCB, GENERAL);
    XtAddCallback(dialog, XmNcancelCallback, dialogCB, QUIT);
    XtAddCallback(dialog, XmNhelpCallback, dialogCB, SPECIFIC);
}

/**************************************************************************/
void query_dialog()
{
    Arg a1[20];
    int ac;

    ac = 0;
    XtSetArg(a1[ac], XmNcancelLabelString,
             XmStringCreateLtoR("No", char_set)); ac++;
    XtSetArg(a1[ac], XmNokLabelString,
             XmStringCreateLtoR("Yes", char_set)); ac++;
    XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) Orange.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Navy.pixel); ac++;
    dialog1 = XmCreateMessageDialog(toplevel, "RETRIEVAL QUERY", a1, ac);
    XtAddCallback(dialog1, XmNokCallback, dialog1CB, YES);
    XtAddCallback(dialog1, XmNcancelCallback, dialog1CB, NO);
    XmUnmanageChild(XmMessageBoxGetChild(dialog1, XmDIALOG_HELP_BUTTON));
}
/**************************************************************************/
void query_dialog_insert()
{
    Arg a1[20];
    int ac;

    ac = 0;
    XtSetArg(a1[ac], XmNcancelLabelString,
             XmStringCreateLtoR("No", char_set)); ac++;
    XtSetArg(a1[ac], XmNokLabelString,
             XmStringCreateLtoR("Yes", char_set)); ac++;
    XtSetArg(a1[ac], XmNmessageAlignment, XmALIGNMENT_CENTER); ac++;
    XtSetArg(a1[ac], XmNbackground, (XtArgVal) Orange.pixel); ac++;
    XtSetArg(a1[ac], XmNforeground, (XtArgVal) Navy.pixel); ac++;
    dialog = XmCreateMessageDialog(toplevel, "INSERTION QUERY", a1, ac);
    XtAddCallback(dialog, XmNokCallback, dialog1CB_INSERT, YES);
void query_diaol_delete()
{
    Arg al[20];
    int ac;

    ac = 0;
    XtSetArg(al[ac], XmNcancelLabelString,
             XmStringCreateLtoR("Record", char_set)); ac++;
    XtSetArg(al[ac], XmNokLabelString,
             XmStringCreateLtoR("Name", char_set)); ac++;
    XtSetArg(al[ac], XmNhelpLabelString,
             XmStringCreateLtoR("No", char_set)); ac++;
    XtSetArg(al[ac], XmNmessegeAlignment,XALIGNMENT_CENTER); ac++;
    XtSetArg(al[ac], XmNbackground,(XArgVal) Orange.pixel); ac++;
           XtSetArg(al[ac], XmNforeground,(XArgVal) Navy.pixel); ac++;
    dial_del = XmCreateMessageDialog(toplevel, "DELETION QUERY", al, ac);
    XtAddCallback(dial_del, XmNokCallback, dialogCB_delete.NAME);
    XtAddCallback(dial_del, XmNcancelCallback, dialogCB_delete.RECORD);
    XtAddCallback(dial_del, XmNhelpCallback, dialogCB_delete.NO);
}

void menu_interface_diaol()
{
    Arg al[20];
    int ac;

    ac = 0;
    XmFontStruct *font=NULL;
    XmFontList fontlist =NULL;
    char *namestring =NULL;

    ac = 0;
    XtSetArg(al[ac], XmNheight,600); ac++;
    XtSetArg(al[ac], XmNwidth,500); ac++;
    XtSetValues(toplevel, al, ac);

    ac = 0;
    form = XmCreateForm(toplevel, "form", al, ac);
    XtManageChild(form);

    ac = 0;
    XtSetArg(al[ac], XmNlabelString,
             XmStringCreate("Windor Software Base For Reuse",char_set)); ac++;
    XtSetArg(al[ac], XmNalignment,XALIGNMENT_CENTER); ac++;
    XtSetArg(al[ac], XmNbackground,(XArgVal) Green.pixel); ac++;
           XtSetArg(al[ac], XmNforeground,(XArgVal) Red.pixel); ac++;
    label = XmCreateLabel(form, "label", al, ac);
    XtManageChild(label);

    namestring = "times*24";
    font = XLoadQueryFont(XtDisplay(label), namestring);
    fontlist=XmFontListCreate(font, char_set);

    ac = 0;
    XtSetArg(al[ac], XmNfontlist,fontlist); ac++;


```c
XmSetValues(label, al, ac);

ac = 0;
  XtSetArg(al[ac], XmNforeground, (XtArgVal) LimeGreen.pixel); ac++;
  menu_bar = XmCreateMenuBar(form, "menu_bar", al, ac);
  XtManageChild(menu_bar);

ac = 0;
  XtSetArg(al[ac], XmNeditMode, XmMULTI_LINE_EDIT); ac++;
  XtSetArg(al[ac], XmNscrollVertical, True); ac++;
  XtSetArg(al[ac], XmNeditable, False); ac++;
  XtSetArg(al[ac], XmNmarginWidth, 10); ac++;
  XtSetArg(al[ac], XmNmarginHeight, 10); ac++;
  XtSetArg(al[ac], XmNbackground, (XtArgVal) Purple.pixel); ac++;
  XtSetArg(al[ac], XmNforeground, (XtArgVal) Wheat.pixel); ac++;
  text2 = XmCreateText(form, "text2", al, ac);
  XtManageChild(text2);
  XtsSetColorSensitive(text2, False);

ac = 0;
  XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_POSITION); ac++;
  XtSetArg(al[ac], XmNbottomPosition, 10); ac++;
  XmSetValues(label, al, ac);

ac = 0;
  XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac], XmNtopWidget, label); ac++;
  XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_POSITION); ac++;
  XtSetArg(al[ac], XmNbottomPosition, 15); ac++;
  XmSetValues(menu_bar, al, ac);

ac = 0;
  XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac], XmNtopWidget, menu_bar); ac++;
  XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
  XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_FORM); ac++;
  XmSetValues(text2, al, ac);
}
```
Program Listing

retrieve_dialog();
insert_dialog();
delete_dialog();
gen1sol_dialog();
specific_sol_dialog();
froze1sol_dialog();
no_specific_solutions();
no_froze_solutions();
no_delete_match();
insert_match();
no_name();
placeWidgets();
placeWidgets_insert();
placeWidgets_delete();
exact_dialog();
version();
setup_browser();
setup_drawA();
no_general_solutions();
no_exact_dialog();
help_retrieve();
help_insert();
about_help();
main_help();
help_delete();
query_dialog();
query_dialog_insert();
query_dialog_delete();
menu_interface_dialog();
create_menus(menu_bar);

XtRealizeWidget(toplevel);
XtAppMainLoop(context);
}

/* .............................................................................
 ............................................................................. END
 ............................................................................. */
Appendix C: User Manual

All interaction with WISER is done through a window-base I/O and graphical user-friendly interface, constructed with the windowing facilities of Motif. This interface was designed so as to allow uniformity among windows and consequently attract new inexperienced users. A number of help menus are also provided with the hope of navigating the user successfully through the required motions necessary to formulate a query and view its results. The system can be invoked by typing WISER at the UNIX shell prompt.

C.1 Main Window

The user starts his/her interaction with the system by first confronting the main window (see Figure C.1) and based upon the action invoked, dialog windows are made available. The main window can be partitioned into two parts, the first can be thought of as consisting of the main menu while the second can be considered as being textual in nature. From the main menu the user can invoke a number of useful dialogs such as the retrieval, insertion, deletion and browse dialogs with which to enter their queries. The textual part is responsible for displaying the source code of various functions as initiated by the user. If the user decide that he/she would like to view the source code of a particular function then all they have to do is click on that function in the various dialogs. This will cause the source code for that function to be displayed in textual part of the main window.

Each of the item in the main menu has a pull-down menu attached to them. So from the File item you can select any of the following options:

1. Save

2. Empty Document
3. **Quit**

From the Retrieve, Insert and Delete items you can chose the *Retrieve, Insert and Delete* dialogs respectively, but other options will be made available in the next version. The *Browse* item contains an *Browse* dialog that is used to set the number of arguments for the browser. The *Version* item gives you information pertaining to version 1.0 of WISER. Finally an on-line help is available by selecting the *Help* item which has two
items in its pull-down menu called About and Help. To perform a selection click on the
LEFT mouse button after positioning the cursor over the desired option.

C.2 Retrieval

To retrieve a component from the software base requires you to formulate a query
in terms of the type of the required function. This is done by following the steps given
below:

1. From WISER's main window (see Figure C.1) you select the button labeled Retrieve
   from its main menu. This will cause the activation of the RETRIEVE_popup dialog,
as shown in Figure C.2.

2. In the RETRIEVE_popup dialog slide the horizontal scroll bar labeled, 'Number
   Of Arguments', to correspond to the number of arguments in the function desired.
   Doing so will allow for a list of arguments to be displayed.

3. Move the mouse so that the cursor is positioned over the first argument in the list.
   Then click on the LEFT mouse button, this will cause the current content of that
   argument to be displayed in the box labelled 'Argument Type:'.

4. Move the cursor to the box labelled, 'Argument Type:', and click on the LEFT
   mouse button.

5. The content of that box, which corresponds to a particular argument, can now be
   edited.

6. When finish editing or entering that argument's type, press the enter key on the
   keyboard.

7. Repeat 3. to 6. for other arguments.

8. If an incorrect type is detected in the list after editing, then correction can be made
   by following 3. to 6. for that particular argument.
9. Move the cursor to the box labelled, 'Result Type:', and click the LEFT mouse button, to allow for editing.

10. Edit or enter the type of the result, for the function desired.

11. There is no need to press the return key in this box, since it is inoperative here.

12. If everything seems correct then press the 'APPLY' button with the LEFT mouse button. This will cause a confirmation dialog (see Figure C.3) to appear. If changes needs to made to the query then select No else select Yes. If No is selected then the RETRIEVE_popup dialog appears and repeat from 2.

13. To reset the RETRIEVE_popup dialog to its original state then press the 'RESET' button with the LEFT mouse button.

14. To terminate the retrieval process press the 'QUIT' button with the LEFT mouse button.

15. On-line help is also available for this dialog by selecting the 'HELP' button.
C.2.1 Various Matches

Once the you chooses the Yes button on the RETRIEVE QUERY_popup dialog, one of two dialogs will appear. One is called EXACT SOLUTION(S)_popup dialog (see Figure C.5), and appears if there is a function whose type forms an exact match with the query type. The other is called SOLUTION_popup dialog (see Figure C.4), and appears if no exact match is found.

From either of these two dialogs, the user can chose to find more specific functions (see Figure C.6), more general functions (see Figure C.7), functions whose types are the same as the query type by freezing an argument (see Figure C.8) or quit. To perform any of these action just position the cursor (use the mouse) over the appropriate button and press the LEFT mouse button.

Finally, to display the source code of a particular function in any of the dialogs above, just position the cursor (use the mouse) over the item in the list containing that
function and its related type, and click the LEFT mouse button. This action will cause
the source code to be displayed in the textual part of the main window (see Figure C.1).

C.3 Insertion

To insert a component into the software base requires you to formulates a query in
terms of the type of that function along with its name and source code. This is done by
following the steps given below:

1. From WISER's main window (see Figure C.1) the user selects the button labeled
   Insert from it main menu. This will cause the activation of the INSERT_popup
dialog, as shown in Figure C.9.
Figure C.6 Specific function dialog

Figure C.7 General function dialog
2. In the INSERT_popup dialog slide the horizontal scroll bar labelled, 'Number Of Arguments', to correspond to the number of arguments in the function being inserted. Doing so will allow for a list of arguments to be displayed.

3. Move the mouse so that, the cursor is positioned over the first argument in the list. Then click on the LEFT mouse button, this will cause the current content of that argument to be displayed in the box labelled 'Argument Type:'

4. Move the cursor to the box labelled, 'Argument Type:', and click on the LEFT mouse button.

5. The content of that box, which corresponds to a particular argument, can now be edited.

6. When finish editing or entering that argument's type, press the enter key on the keyboard.

7. Repeat 3. to 6. for other arguments.
8. If an incorrect type is detected in the list after editing, then correction can be made by following 3. to 6. for that particular argument.

9. Move the cursor to the box labelled 'Result Type:', and click the LEFT mouse button, to allow for editing.

10. Edit or enter the type of the result, for the function being inserted.

11. There is no need to press the return key in this box, since it is inoperative here.

12. Move the cursor to the box labelled 'Function Name:', and click the LEFT mouse button, to allow for editing.

13. Edit or enter the name of the function being inserted.

14. If everything seems correct then press the **APPLY** button with the LEFT mouse button. This will cause a confirmation dialog (see Figure C.10) to appear. If changes needs to made to the query then select No else select Yes. If No is selected
then the INSERT_popup dialog appears and repeat from 2. else the function is inserted unless it already exists, in which case Figure C.11 appears.

15. To reset the INSERT_popup dialog to its original state then press the 'RESET' button with the LEFT mouse button.

16. To terminate the insertion process press the 'QUIT' button with the LEFT mouse button.

17. On-line help is also available for this dialog by selecting the 'HELP' button.

C.4 Deletion

To delete a component from the software base requires you to formulate a query in terms of the type of that function along with its name. This is done by following the steps given below:

1. From WISER's main window (see Figure C.1) the user selects the button labeled Delete from it main menu. This will cause the activation of the DELETE_popup dialog, as shown in Figure C.12.
2. In the DELETE_popup dialog slide the horizontal scroll bar labelled "Number Of Arguments", to correspond to the number of arguments in the function being deleted. Doing so will allow for a list of arguments to be displayed.

3. Move the mouse so that the cursor is positioned over the first argument in the list. Then click on the LEFT mouse button, this will cause the current content of that argument to be displayed in the box labelled "Argument Type:"

4. Move the cursor to the box labelled "Argument Type:”, and click on the LEFT mouse button.

5. The content of that box, which corresponds to a particular argument, can now be edited.

6. When finish editing or entering that argument’s type, press the enter key on the keyboard.
7. Repeat 3. to 6. for other arguments.

8. If an incorrect type is detected in the list after editing, then correction can be made by following 3. to 6. for that particular argument.

9. Move the cursor to the box labelled, 'Result Type:', and click the LEFT mouse button, to allow for editing.

10. Edit or enter the type of the result, for the function being deleted.

11. There is no need to press the return key in this box, since it is inoperative here.

12. Move the cursor to the box labelled 'Function Name:', and click the LEFT mouse button, to allow for editing.

13. Edit or enter the name of the function being deleted.

14. If everything seems correct then press the 'APPLY' button with the LEFT mouse button. This will cause a confirmation dialog (see Figure C.13) to appear. If changes needs to made to the query then select No else select Name (Record is inoperative). If No is selected then the DELETE_popup dialog appears and repeat from 2. Relevant error messages are also available (see Figures C.14 and C.15).

15. To reset the DELETE_popup dialog to its original state then press the 'RESET' button with the LEFT mouse button.

16. To terminate the deletion process press the 'QUIT' button with the LEFT mouse button.

17. On-line help is also available for this dialog by selecting the 'HELP' button.
C.5 Browser

To browse the structured software base requires you to formulate a query, in terms of the number of arguments, so as to determine which part of the structure to start with. This is done by following the steps given below:

1. From WISER's main window (see Figure C.1) you select the button labeled Browse from it main menu. This will cause the activation of the ARG_BROWSER_popup dialog, as shown in Figure C.16.

2. From the ARG_BROWSER_popup dialog position the cursor over the slider on the scroll bar (use the mouse to do this). Now press down on the LEFT mouse
button and move the slider until the number shown is the desired one. At this point release the button. If everything is okay then press OK, if not then repeat the same procedure. To perform a selection click on the LEFT mouse button after positioning the cursor over the appropriate button/selection.

3. By selecting the OK button will cause the BROWSER_popup dialog to appear (see Figure C.17). This dialog consists of three buttons and a drawing area to show the structure in stages.

4. The browser displays all the parents, children and extra-argument type of a particular type. It also displays all the functions associated with that particular type if any exists.

5. To navigate the structure just place the cursor on any of the types and click on the LEFT mouse button. This will cause that type to be placed in the middle with its associated functions, and its parents, children and extra-argument type will be made visible. This procedure can then be repeated until you are satisfied. Note: you dictate the navigation path.

6. The Argument button is selected if you want to view some other part of the structured software base that is not readily accessible from the present location, as in terms of trying to view the structure where the number of argument are less than the those showing.

7. The Help button provides on-line help.

8. When finish select Quit to kill the BROWSE_popup dialog.
Figure C.17 Browser window

**Table:**

<table>
<thead>
<tr>
<th>Parent Level</th>
<th>Freeze Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)→(b)</td>
<td>(c)→(d)</td>
</tr>
<tr>
<td>(b)→(c)</td>
<td>(d)→(a)</td>
</tr>
<tr>
<td>(c)→(d)</td>
<td>(a)→(b)</td>
</tr>
</tbody>
</table>

**Children Level:**

<table>
<thead>
<tr>
<th>(char)&lt;→(char)</th>
<th>(char)&lt;→(char)</th>
<th>(num)&lt;→(num)</th>
<th>(num)&lt;→(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(char,b)&lt;→(char,b)</td>
<td>(char,b)&lt;→(char,b)</td>
<td>(num)&lt;→(num)</td>
<td>(num)&lt;→(num)</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


BIBLIOGRAPHY


VITA AUCTORIS

Deoraj Ramjisingh was born in 1969 in Georgetown Guyana. He graduated from Queens College in 1987. From there he went on to the University of Windsor where he obtained a B. Sc. in Computer Science in 1993. He is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in the Spring of 1995.