Amorphous Slicing Of C-Programs

Project Report

By

Mir Abubakr Shahdad
Student Number: 0740009

Supervision & Guidance
By:

Prof Mark Harman

CREST, DCS
Kings College London

2nd September 2008
Acknowledgements:

It gives me immense pleasure to express my thanks and appreciation to all those who helped me in the completion of this project work. Above all I thank Almighty ALLAH for his blessings and strength that he bestowed upon me throughout the course of my work. I would like to express my gratitude to the following people who provided constant help and support throughout this project.

The first name to start with in my mind comes Prof. Mark Harman, who constantly supported me and was always there to extend his support and expert consultation. Also, he made me to complete critical tasks of the project much ahead of the deadlines and hence at last I found that there was enough room and time for me to check out for any corrections, amendments and errors. Prof. Mark also supervised in such a way that I was able to perform at the best level during the entire course of this project.

My second marker, Dr. Iman Poermono at Kings DCS-PALABS also helped me a lot by suggesting .NET at the presentation and also helping me make the UML2 Diagram for the project.

One of my other project guides George also deserves a word of mention here who always helped me carve out my technical skills in general.

My friend Malik Ali at King’s who usually helped me resolve particular issues in this project and was always came up with his good suggestions for me. My friend Asif from Musky Software Solutions California also proved to be a great inspiration to take this project to a satisfactory completion.

My lifetime friends Kausar and Afshan, Saima, Asim, Nazia and Shahid who always prompted me to do my best in this project and in my MSc. They always kept boosting my energy and confidence for making amendments in the work.

Last but not the least are my parents, my sister, brother, brother In-law Adil, Ahmad who have always been supporting me in all the positive things I did in my life.

Thanks to All.

Abubakr Shahdad
Abstract

It is an accepted fact that software testing has been allotted more than half of total project cost as well as budget in large scale enterprise software development life cycle. As, testing professionals can never be sure as to whether a software product is bug free or not, but to a large extent it can be made sure that a product is nearly error free by testing it following a set of established methods of software testing and measurement.

The most important problem in software testing which the professionals and companies face nowadays is the existence of a large number of test cases which need to be covered. The problem is further amplified by two factors, one is the program size and other is the absence of automated testing tools to some extent.

While attempts are being made to solve the problem of exhaustive test cases by adopting the evolutionary testing and other methods by which the best test cases are followed. On the other hand, the method of program slicing is adopted to decrease the size of the program is decreases. It has been agreed to some extent that surely a program slice is smaller than the original program size. But still a great amount of research is needed in this field to establish results. Also, we need more automated and comprehensive slicing tools for the programs.

A question which still remains unanswered to a large extent is as follows: Is an amorphous slice smaller than syntax preserved slice?. In other words, is amorphous slice in any way any more useful than its counterpart of syntax preserved slice. The aim of this project is to develop an amorphous slicer for C programs and then analyse a set of C programs for their syntax preserved slice and then their amorphous slice. Hence at last we will be able to demonstrate the relative advantages or disadvantages of using an amorphous slicer over syntax preserved slicer. In either case, a research question addressed by software research community will be answered in context of a limited syntax of C over a group of sample C programs.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>1. Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Evaluation / Project Aim &amp; Problem Definition</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Suitability Of .NET for this project</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Background</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Program Slicing, Debugging &amp; Amorphous Slicing</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Previous Work &amp; Papers</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Benefits by this Project</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Slicing Algorithms- Push Transformation</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Push Transformation with Loop Statements</td>
<td>11</td>
</tr>
<tr>
<td>2.6 Push Transformation with Conditional Statements</td>
<td>12</td>
</tr>
<tr>
<td>2.7 Slicing Algorithms- Kill Assignments</td>
<td>14</td>
</tr>
<tr>
<td>2.8 Slicing Algorithms- Loop Squash Algorithm</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Design</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 UML-2 Component Diagram Of System</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Tools Used In This Project</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. Data ( Program Samples ) Testing</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Tested Programs</td>
<td>28</td>
</tr>
<tr>
<td>4.2 Comparison Of Slices ( data ), Criteria(v,n)</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Limitations Of system</td>
<td>28</td>
</tr>
<tr>
<td>4.4 What portion of C was covered</td>
<td>29</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5. Analysis &amp; Conclusion</td>
<td>30</td>
</tr>
<tr>
<td>5.1 Graphs Demonstrating the studies</td>
<td>30</td>
</tr>
<tr>
<td>5.2 Conclusion &amp; Extended Implication of results</td>
<td>31</td>
</tr>
<tr>
<td>6. Project Planning &amp; Phases</td>
<td>33</td>
</tr>
<tr>
<td>6.1 GANTT chart</td>
<td>33</td>
</tr>
<tr>
<td>6.2 Main phases of the project</td>
<td>36</td>
</tr>
<tr>
<td>7. Future Work</td>
<td>37</td>
</tr>
<tr>
<td>7.2 Where this project was left</td>
<td>37</td>
</tr>
<tr>
<td>7.3 Future Work</td>
<td>37</td>
</tr>
<tr>
<td>Glossary &amp; References</td>
<td>39</td>
</tr>
<tr>
<td>Glossary</td>
<td>39</td>
</tr>
<tr>
<td>References &amp; Bibliography</td>
<td>40</td>
</tr>
<tr>
<td>Appendix</td>
<td>42</td>
</tr>
<tr>
<td>Appendix A – Interface Of project</td>
<td>42</td>
</tr>
<tr>
<td>Appendix B - Tested Programs &amp; Slices Obtained</td>
<td>45</td>
</tr>
<tr>
<td>Appendix C – Source code of Project</td>
<td>52</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Size of a program has been the biggest issue that both the manual and automated testing professionals have been facing till now and are facing it currently. Localized identifying and fixing of errors is the most important way to success for making a quality compliant and error free software. Software slicing is one of the decomposition techniques used in software testing and debugging. It is used to determine the sub sets of statements relevant to a computation of interest. Its output is the portion of program (slice) which directly or indirectly contributes to the value of a given variable at a particular position (line of code).

It is usually predicted that the size and complexity of a program slice are less than that of the original program. Thus, the resulting slice can be easily debugged and in that way the error can be located and fixed easily. Program slicing can be particularly useful, when the size of original program is very big, and localization of error is very hard in such cases leading to wastage of potential and time.

The best use of slicing can be done when the particular type of slicing is applied at different situations. Mark Weiser invented slicing in his PhD thesis in 1979. Weiser claimed that slicing is what the programmers do any way while debugging. For example, if an error regarding a variable is to be located, programmers first of all check the lines of code which affect that variable, usually working backwards from the point of error. They can justify their actions by arguing that why those lines of code should be considered which do not contribute to that error.

1.1 Problem Definition / Evaluation

Firstly, through this project we achieved an amorphous slicer for C programs. Secondly, the evaluation criterion for this project is to conclude whether an amorphous slice is smaller than its counterpart syntax preserved slicer and the original program itself. And also we will present the ratio of sizes hence presenting the downsize percentage ratio. The overall
efficiency comparison will also be conducted in the conclusion part of this paper. In addition to this, the scenarios best suited to apply syntax or amorphous slicer will be discussed.

### 1.2 Suitability of .NET for this project

This project is primarily composed of a lexical analyzer followed by other source code analysis tools. Thus, as is evident from the requirements that a strong string processing was to be implemented. The choice of better tools would result in a rapid and a high quality result. At the moment of decisions in project design, two tools were suitable - Java and .NET. As I already had a good knowledge of Microsoft based technologies, so choosing .NET was the right decision at the right time.

A fast tool was needed for this project. Besides that, qualities like sub-string processing, pattern matching and character extraction were expected from the tool of choice. Regarding the string processing in .NET, some of the features (syntax) worth mention here are:-

- `String.concat`
- `String.contains`
- “Like” in string processing
- `String.Empty`
- Regular expressions
- `String.indexOf`
- `String.LastIndexOf`
- `String.Insert`
- `String.Join`
- `String.Length`
- `String.Trim/TrimStart/TrimEnd`
- `String.Split`
- `String.Replace`
- `String.StartsWith`
- `String.Char`
- `String.padleft/padright`
CHAPTER 2

BACKGROUND

Program slicing is an efficient technique used in debugging and the error removal.

2.1 Program Slicing, debugging & Amorphous Slicing

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another. [11]

Debugging is, in general, a cumbersome and tiring task. The debugging skill of the programmer is probably the biggest factor in the ability to debug a problem, but the difficulty of software debugging varies greatly with the programming language used and the available tools, such as debuggers. Debuggers are software tools which enable the programmer to monitor the execution of a program, stop it, re-start it, set breakpoints, change values in memory and even, in some cases, go back in time. The term debugger can also refer to the person who is doing the debugging. The test debug cycle is a process in which once we develop the indented part (or whole) program, then we start testing the program for errors, and we continue fixing and debugging the program till the program doesn’t show errors on testing. Slicing is an extended practice of what the programmers usually so while they debug and fix a program. [11]

Program slicing is a decomposition technique aimed at determining the subset of statements relevant to a computation of interest. It provides the program statements which directly or indirectly contribute to the value of a given variable x at a given statement n. Mark Weiser invented slicing in his PhD thesis in 1979. Weiser claimed this is what programmers do anyway when debugging. [12, 13]
Slicing has many applications:

- Debugging
- Testing
- Parallelization
- Program Integration
- Aid in understanding a program
- Maintenance
- Measurement by cohesion
- Program Comprehension
- Quality Assurance
- Re-Engineering

Additionally, some of the emerging applications of program slicing are:

- Security
- Virus Removal
- Modal checking
- Clone detection
- Data mining
- Understanding business models

There are many different variants of slicing; among them are static slicing, dynamic slicing, forward slicing, backward slicing, condition or quasi-static slicing, syntax-preserving slicing, and amorphous slicing. It is possible to split the slicing paradigms up into two main categories: semantic and syntactic. The semantic paradigms include the static, dynamic, and conditioned while the syntactic paradigms include syntax-preserving and amorphous slicing. The semantic and syntactic elements are two important aspects of slicing. The semantic element describes what part of the program is to be preserved. With the syntactic element, there are two possible types of slicing: 1) the program’s original syntax can be preserved, removing sections of the program that have no affect on the
Amorphous Slicing of C-Programs

semantics of interest, and 2) syntax transformations take place, which preserve the semantic detail of the program. [2]

Hence, we can have following sub categories of slicing [14]

- **Static Slicing**

  The original slicing paradigm, static slicing is a simple and commonly used tool for constructing program slices. All other forms of slicing can be considered as an extension to static slicing. A program slice is produced by deleting the statements that have no semantic significance to the set of variables, know as the slice set, at the chosen point in the program. Thus, given a set a variables V at a specific point of interest n, a slice can be constructed for the variables in V at point n. After selecting a slicing criterion, we have a choice of two forms of slice: a backward slice or a forward slice. The difference between the two slices is that a backward slice contains the program statements that have an effect on the slicing criterion, whereas a forward slice contains the program statements that are affected by the slicing criterion. Static slices tend to be rather large, especially when highly cohesive programs are sliced. [14]

  This is because high cohesion occurs in programs where the computation performed on each variable is dependant on many other variables. However, slices constructed by static slicing do simplify the program, which helps in debugging activities.

- **Dynamic Slicing**

  Static slicing constructs slices for variables at a specified point in the program of interest. Some of those variables have a possible range of inputs that determine, in many cases, the behaviour or the output of a program. During the execution of a program, it may be that a value inputted causes some unexpected result, either in the form of incorrect values being outputted or program termination or both. [14]

  A dynamic slice takes advantage of the available information about the input sequence supplied to a program at a specific execution. This dynamic information is
Amorphous Slicing of C-Programs

extremely valuable when considering the presence of faults or bugs in a program. In the event of a failure occurring, the input to a program is available and can be used to locate the source of that failure. The slice will then only contain those statements that could have caused the failure within a specific execution of interest. The static slice criterion together with the dynamic information (the sequence of input values for a variable), make up the ‘dynamic slicing criterion’, so given a variable v at a specific point of interest n on an input i, we can construct a dynamic slice. It is clear that dynamic slices are superior to static slices when the application is debugging, but this does not mean that we no longer need static slicing. Other applications, such as code re-use, require slices to be consistent for every possible execution. Since both slicing techniques have their advantages and disadvantages according to their applications, there is a trade-off between the static and dynamic paradigms, with static slices being larger but catering for all possible executions and dynamic slices being smaller but only catering for a single input. [14]

- **Conditioned Slicing**

  So far, we have talked about two ways of constructing a slice, with one (static) providing no information about the input to the program, and the other (dynamic) providing specific information about the input. Conditioned slicing bridges the gap between the two previous paradigms by using a Boolean condition to specify a range of inputs rather than precise values. The conditioned approach has been applied successfully to problems related to program understanding. Slicing with respect to conditions tells us a lot about how a program behaves under those conditions. Each conditioned slice contains the statements that capture various aspects of a program’s behaviour when the conditions stated are satisfied. [14]

- **Amorphous Slicing**

  The static, dynamic, and conditioned approaches to slicing are all ‘syntax-preserving’. By this we mean that the syntax of the slices is the same as that of the original program from which the slices were constructed. Therefore, these slices could be thought of as syntactic subsets of the original program. The only transformation that may have taken
place when constructing the slices would have been the deletion of the statements that were irrelevant to the slice criterion. Amorphous slicing transforms the program syntactically to simplify it, preserving the semantic behaviour with respect to the slicing criterion. An amorphous slice is never larger than its syntax-preserving equivalent and is often significantly smaller. Because amorphous slicing has the ability to extract the semantics of interest from a program by transforming the syntax, it can aid program comprehension, analysis, and re-use. [14]

2.2 Previous Work And papers, Review Of Literature

The primary paper which introduces the amorphous slicing is the paper published by Prof. Mark Harman and Sabastian Danicic. A theoretical framework of program projections is introduced in that paper. Some work on amorphous at DCS King’s College London was done by Kanagasabhai Sriskanthaval as his MSc project work under Prof Mark Harman. Sris used TXL to achieve his targets.

Traditional program slicing simplifies a program by removing program components that do not affect a computation of interest. The resulting slice captures a projection of semantics of the original program. Figure below illustrates syntax preserving program slicing with a simple example of a program and its slices. [8, 2, 4]

<table>
<thead>
<tr>
<th>Original Program</th>
<th>Slice (pe,9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: scanf(&quot;%d&quot;,&amp;h);</td>
<td>1: scanf(&quot;%d&quot;,&amp;h);</td>
</tr>
<tr>
<td>2: pe=0;</td>
<td>2: sum=0;</td>
</tr>
<tr>
<td>3: pt=1;</td>
<td>3:</td>
</tr>
<tr>
<td>4: while (h&gt;0);</td>
<td>4: while (h&gt;0);</td>
</tr>
<tr>
<td>5: {</td>
<td>5: {</td>
</tr>
<tr>
<td>6: pe=pe+h;</td>
<td>6: pe=pe+n;</td>
</tr>
<tr>
<td>7: pt=t*h;</td>
<td>7:</td>
</tr>
<tr>
<td>8: h=h-1;</td>
<td>8: h=h-1;</td>
</tr>
<tr>
<td>9: }</td>
<td>9: }</td>
</tr>
</tbody>
</table>

Table 2.1 – Syntax preserving slice of program example
Amorphous slicing is a variation of traditional slicing in which the syntactic requirement is dropped while the semantic requirement is retained. For a slice taken with respect to \( v \) at a statement \( s \), amorphous slicing preserves the effect of the original program on the values of variable \( x \) at \( s \), but places no restriction upon the syntax of the slice. [8, 2, 4]

For example, let us again take an example as shown below. The fragment was written with the intention that the biggest would be assigned the largest value in the 20-element array \( a \), and that the variable average would be assigned the average of the elements of \( a \). However, the fragment contains a bug which affects the variables sum and average, but not the variable biggest. [8, 2, 4]

\[
\begin{align*}
\text{For}(i=0, \text{sum}=a[0], \text{biggest}=\text{sum}; i<19; \text{sum}=a[++i]) \\
\text{If}(a[i]>\text{biggest}) \\
\text{Biggest}=a[i+1]; \\
\text{Average}=\text{sum}/20;
\end{align*}
\]

Table 2.2 – Example program to illustrate amorphous slicing

To illustrate amorphous slicing, the variables biggest and average will be analyzed using both traditional syntax-preserving slicing and amorphous slicing. The syntax preserved slice on the final value of biggest consists of all but the latest statement. This does not help make clear the correctness of the computation of biggest. However, the amorphous slice, constructed using transformations such as a loop unrolling (which has changed the loop bounds) is the following:

\[
\begin{align*}
\text{For}(i=1, \text{biggest}=a[0]; i<20; ++i) \\
\text{If}(a[i]>\text{biggest}) \\
\text{Biggest}=a[i];
\end{align*}
\]

Table 2.3 Example of amorphous slice

From the amorphous slice it is a little bit clearer that the computation on biggest is correct. The syntax-preserving slice on the final value of average contains all but the if statement
from the original program. Once again, this simplification is a relatively weak aid to comprehension. However, the amorphous slice goes much further. [8, 2, 4]

\[ \text{Average} = a[19]/20; \]

### 2.3 Benefits By This Project

There are manifold benefits by this project. Firstly, we will achieve an amorphous slicer. Secondly, we got a solid comparison result showing that amorphous slice can in majority of cases be much smaller in size than either the original program or the syntax preserved slice counterpart.

### 2.4 Push Transformation

Push transformation is an amorphous slicing technique to down size the line count of the slice, by way of pushing assignment expression to variables wherever possible, to its references down the program. For an example consider this simple C program. [8]

```c
1 void main(){
2     int a, b;
3     a = 3;
4     b = 2 * a + 4;
5     printf("%d", b);
6 }
```

In this example, in line 3 a is assigned to a value of 3; and line 4 assigns an expression to b which depends on a. A push transformation replace the instances of a down the program with the expression a is assigned to, which is ’3’. Therefore the push transformation replace line 4 with \( b = 2*3 + 4 \). Further the assignment to b in line 4 is pushed down in statement 5 to get printf(“%d”, 2*3 + 4). By applying the expression simplifying algorithm discussed in
section 3.3 in this report, \(2 \times 3 + 4\) is simplified to 10, and the expected output will be as shown below in step by step. [8]

```
void main()
{
    int a, b;
    a = 3;
    b = 2 \times 3 + 4;
    printf("%d", b);
}
```

Table 2.2

However, a reassignment to either the pushing variable or any variable in the replacement expression will result the push transformation further down the program incorrect. The example shown in Table 2.3 explains this scenario better.

```
1 void main()
2 {
3    int a, b, c;
4    scanf("%d", c);
5    a = 3;
6    b = 2 \times c + 4;
7    printf("%d", a);
8    a = 7;
9    c = 3 \times a;
10   printf("%d", b);
11   printf("%d", c);
12 }
```

Table 2.3

The assignment statement at line 4, \(a = 3\), can be pushed down until the value of \(i\) has been reassigned. In the example line 4 can be pushed down up to line 7, replacing statement 6 with `printf("%d", 3);`. Further, assignment statement at line 7 can be pushed down until the end of the program, as there is no reassignment to \(i\) after line 6. In case of statement \(b = 2 \times c + 4\) in line 5, the expression \(2 \times c + 4\), can be pushed down the program until a reassignment to either \(j\) or any variables in the replacement expression \(2 \times c + 4\), which is \(\{c\}\). Since there is a reassignment to \(c\) in line 8, line 5 can be pushed down up to line 8. It is incorrect to push the expression \(2 \times c + 4\) to the instance of \(b\).
Amorphous Slicing of C-Programs

at printf(“%d”, b); in line 9; because, the value of b at line 5 is computed with the
value of c input by the user at line 3, this is the value of b that is expected to appear in line
9. On the other hand, if the expression 2*c + 4 at line 5 is pushed to line 9 to get printf
(“%d”, 2*c + 4);, the value of the expression in the printf statement is evaluated
based on the value of c that has been reassigned to c=3*a; at line 8, which is incorrect.
The expected push transformed slice of the program listed in Table 2.3 is as shown in Table
2.4. [8]

```
1 void main(){
2   int a, b, c;
3   scanf("%d", c);
4   b = c;
5   j = 2 * c + 4;
6   printf("%d", 3);
7   c = 3 * 7;
8   printf("%d", b);
9   printf("%d", 3 * 7);
10  }
```

Table 2.4

2.5 Push Transformation With Loop Statements

When it happens to any reassignment to a variable inside a loop, similar to conditional
reassignment, the previous assignment can't be pushed further down the loop statement. In
addition, it is incorrect to push down the previous assignment inside the loop, or the looping
invariant, considering the fact that a reassignment inside a loop, affect the references to it in
expressions anywhere in the loop and the looping invariant. [8]

```
1 void main() {
2   int a, b, c;
3   scanf("%d", c);
4   b = c;
5   while(b > 0){
6     printf("Before: %d", b);
7     b = b - 2;
8     printf("After: %d", b);
9   }
10   printf("Final: %d", b);
11  }
```

Table 2.5
Consider the sample program in Table 2.5. The assignment to \( b \) in line 4, is reassigned at line 7, inside the while loop. Since statements 5, 6, 7, 8 (the whole body and the looping conditional expression of the while loop) may get executed after reassignment to \( b \) at line 7, obviously, assignment at line 4 can be pushed only up to line 4. Further, an assignment to a variable inside a while loop, can be pushed down the statements within the scope of the loop, as it is guaranteed that they are executed in order inside the loop. However as a special case, if an assignment expression having reference to the assigning variable itself in an assignment statement (the case in line 7), those statements are not considered for push transformation because of its complicated behaviour. To see the problem of pushing an assignment expressions of this kind consider the program in table 2.5 again. The assignment expression in \( b = b - 2; \) can be pushed into statement in line 8, provided assignment statement at line 7 is eliminated after being pushed. On the other hand, If line 7 is not eliminated the segment of the program appear as follows, will print the value at line 8 as \( b - 4 \) as it is reduced by 2 in line 7 and line 8 also prints (b -2), which is incorrect.

| 7 b = b - 2;                |
| 8 printf("After: \%d", b-2); |

Table 2.6

It is also not possible to eliminate the assignment statement at line 7, after being pushed. Because the value of \( b \) is being used at lines 5, 6 and 10, which have dependency with line 7. [8, 2, 4]

### 2.6 Push Transformation With Conditional Statements

As discussed before, an assignment to a variable can be pushed down the program until another statement reassigns the variable to a different value. This statement holds true even for a conditional reassignment. However pushing down the reassigned value further down is restricted within the scope of the conditional statement itself. Any reference to this variable further down the conditional reassignment can't be replaced with the conditionally assigned
value, as it can’t be determined at compile time whether the conditional reassignment will be executed. [8]

```c
1 void main(){
2     int a, b, c, d;
3     scanf("%d", c);
4     a = c;
5     b = 2;
6     printf("%d, %d", a, b);
7     scanf("%d", c);
8     if( c > 0){
9         a = 2 * c;
10        printf("%d, %d", a, b);
11    }
12    printf("%d, %d", a, b);
13 }
```

Consider the sample program in Table 2.5. At line 4, \(a\) is assigned to \(d\), which can be pushed down until it get reassigned in line 9. Hence line 6 will get replaced with `printf("%d, %d", a, b);`. Assignment to \(b\) in line 5 can be pushed down until the end of the program, as there is no reassignment to \(b\) down the program. The conditional assignment to \(a\) in line 9, can be pushed down only within the scope of the if statement. Hence line 10 will get replaced with `printf("%d, %d", 2*d, b);`. However at line 12, it is only at run time, the value of \(a\) can be determined, as the conditional expression at 8 may be true, which leads to the reassignment \(a = 2 * d;\), otherwise the previous assignment at line 4, \(a = d;\) remains valid. That is the value of \(a\) at line 12 can't be determined by statically analyzing the program. The expected push transformed output of the above program may be similar to the one shown in Table 2.6. [8]

```c
1 void main(){
2     int a, b, c, d;
3     scanf("%d", d);
4     a = d;
5     b = 2;
6     printf("%d, %d", d, 2);
7     scanf("%d", c);
8     if( c > 0){
9         i = 2 * d;
10        printf("%d, %d", 2*d, 2);
11    }
12    printf("%d, %d", a, b);
13 }
```
Table 2.8

11 }
12 printf("%d, %d", a, 2);
13 }

.NET Implementation of Push Transformation

.NET implementation can be simply done by implementing the algorithm of push transformation. For example, when we encounter a variable declaration or reassignment which can be resolved into a value, then we override our variables set.

For example for the following statement:

```c
Int a =9;
```

The .NET code will act as follows:

```c
Varcounter=varcounter+1
Var_name(varcounter)="a"
Var_value(varcounter)=9
Var_type(varcounter)="Integer"
```

Table 2.9 - .NET implementation example of PT

2.7 **Kill Assignments (Eliminating Pushed Assignments)**

If an assignment expression is pushed down to all the statements up to the reassign statement, and the reassign statement is in the same scope of the program, it is safe to eliminate the assignment expression after being pushed down. However, if the reassignment is a conditional reassignment or inside a loop statement, it is not guaranteed that the reassignment will be executed. Further in case of a reassignment inside a loop statement, the references to the assigning variable may not get replaced in all places. Therefore it is not possible to eliminate the assignment statements all the time after being pushed down. This algorithm does not eliminate any of the assignment expression from the program. However, those statements that can be eliminated, will be identified as a redundant assignment, and eliminated. [8]
.NET Implementation of Kill Assignment

Our .NET implementation of kill assignment has been made simple and the code has been made in comment form as follows: For example, when a line results in an undisputed and resolved re assigned variable on LHS of assignment statement, then the line is not again printed back to the out put slice, there we indicate by commenting the “print file line” of code or the “display line” of code.

The flow chart -2.1 shows a simple implementation.
Figure 2.1- Implementation of Kill Assignment for a Possible Case.
2.8 Loop Squash Algorithm

The loop squash algorithm is one of the most efficient algorithms, where very small slice can be achieved. It mainly includes squashing a loop structure to as small statement as possible. This method completely eliminates the loop statement into either and if else statement or a simple assignment. However, the exact output depends on variable dependency and the context of amorphous slice. Some cases can be discussed in this section.

If a loop has variables with no user dependency (variables are not scanf statement dependent above that loop), then the loop statement can be directly converted into an if else statement by following the iterations. That if else statement can then be converted into a simple assignment by substituting the values of variables.[1]

However, if a loop has some variable dependent on user input before the loop, then the loop can be again squashed down to an if-else statement. Further to this if-else, the final statement achieved depends on the context of slicing. If we want to achieve a conditioned slice, assuming the values of variables to be inputted by user then we can achieve assignment statements from this if-else statement depending on the particular range of values assumed. But, if we don’t want a conditioned slice, then we end up having an if-else statement resulting from the loop statement.[1]

We will present an example to demonstrate the loop squash algorithm more clearly.

Consider the program below to calculate the gravity of an object. [1]

```c
x = d/2;  
y = h/2;  
s = d*h;  
i = 1;  
while (i <= n) 
{ 
    s0 = s;  
    s = s+d*h+i*h0;  
    x = x+(1-s0/s)*(i*d-x);  
    y = y+(1-s0/s)*(h+i*h0/2-y);  
    i = i+1;  
}  
gravity = density*S;
```

Table 2.10 – Segment of original program –Po
Amorphous Slicing of C-Programs

The algorithm starts by detecting a loop induction variable. Provided it is found, it is used as parameter to the loop pattern matching. If successful, both the detection of the induction variable and loop conversion is achievable. [1]

\[
s = d*h; \\
i = 1; \\
\text{while} \ (i <= n) \\
\{ \\
s = s+d*h+i*h0; \\
i = i+1; \\
\}
\]

\[
gravity = density*S;
\]

Table 2.11-p1 Syntax preserved slice of fragment w.r.t. gravity at the end of Po

Squashing combines the use of induction variables and transformation. By contrast, in the literature on source–to–source loop transformation, the transformations preserve loop structures, i.e., such transformations are from loop(s) to loop(s). Finally, the amorphous slicing algorithm described is capable of performing a rather limited form of squashing. [1]

\[
s = d*h; \\
i = 1; \\
\text{if} \ (i <= n) \ { \\
s = s+d*h+i*h0; \\
i = i+n; \\
}\]

\[
gravity := density*S;
\]

Table 2.12 –p2 - Loop Squash Algorithm applied to p1;

The transformation, referred to as loop–induction variable elimination, looks for sub graphs of a program’s System Dependence Graph that represent while loops only two statements. The first statement updates the induction variable. The second updates a Boolean variable and is of the form “\(S = S \&\& i >= 0 \&\& i < n\)”. This rule arises from work on finding array bound violations where the array has been declared to have size \(n\). The second statement is tracking safety thus far in the variable \(S\). [1]
gravity = density * (h0*(n-1)^2 + n*d*h);
.Net Implementation of Loop squash Algorithms

The .Net implementation of Loop squash Algorithm was done using stmtline variable tracker. In this system, a structure of statements was used which stored the statements of the loop itself.

```csharp
While (Not (linein.Contains("")))
    linein = oRead.ReadLine()
    If Not linein.Contains("")) Then
        kk = ""
    End If
    For k = 1 To linein.Length
        kk = kk & " "
    Next
    If kk <> linein Then
        If linein <> "" Then
            i = i + 1
            While (linein.StartsWith(" "))
                linein = linein.Substring(1)
            End While
        End If
    End If
    stmtline(whileno) = linein
    whileno = whileno + 1
End While
```

Table 2.14 – Loop Squash Algorithm Initialization

After the above has been implemented, the lines of statements are executed repeatedly following the condition.

```csharp
While (varvalue(temppos1) < tmp2)
    ' MsgBox(varvalue(temppos1) & " -----" & tmp2 & vbCrLf)
    ' this part of the program deals with all the code without declarations
    ' and scan fs and print fs ( mainly re initialisations
    ' and mainly with the computations and re-computations
    ' let us iterate over the variables in the program
```
For hh = 0 To whileno - 1
linein = stmtline(hh)

Table 2.15 – Loop Squash Algorithm Statement Execution

CHAPTER 3

DESIGN

In the second phase of this project many modules were designed. However, this system didn’t follow a typical OOP’s strategy completely. A procedural approach was used in which either functions were used or inline coding was followed in the program itself.

3.1 UML2 COMPONENT DIAGRAM OF SYSTEM

To demonstrate the system, we use UML2 component diagram for system. A UML 2 deployment diagram depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes. In other words, deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another. You want to create a deployment diagram for applications that are deployed to several machines, for example a point-of-sales application running on a thin-client network computer which interacts with several internal servers behind your corporate firewall or a customer service system deployed using web services architecture such as Microsoft’s .NET. Deployment diagrams can also be created to explore the architecture of embedded systems, showing how the hardware and software components work together. In short, you may want to consider creating a deployment diagram for all but the most trivial of systems. [3, 15]

A detailed UML2 component diagram is shown in the figure 3.1. [3, 15]
FIGURE 3.1 – UML2 COMPONENT DIAGRAM OF SYSTEM.
**SYNTAX PRESERVED SLICER:** These are the two main phases of this project. The first phase is achieved by utilizing R&D tool called Sprite. The syntax preserved slicer component takes a C program as an input and outputs a C program syntax directed slice version of the original program with respect to the slice criterion \((v, n)\), where \(v\) is a variable in a program and the \(n\) is the line number in the program.

**AMORPHOUS SLICER:** This phase of the project is built using VB.NET. It receives its input as the output of phase 1. In this component, the syntax preserved slice is taken as input to yield the amorphous slice as output. Amorphous slicer component which is the VB.NET portion of the project uses the services of various other components.

- **FILE READER & WRITER:** This piece of code is used to read the input file (syntax preserved slice) as output of the syntax preserved tool. The other function of this component is to output the compressed file as amorphous slice. Besides, achieving an output amorphous slicer file, a text area is also provided on the form to show the amorphous slice.

Following code is example of how files could be read and opened up:

```csharp
Dim oFile As System.IO.File
Dim oRead As System.IO.StreamReader
oRead = oFile.OpenText(Form1.path)
```

- **LINE READER:** It is the piece of code that reads individual lines of code it is implemented as following piece of code:

  ```csharp
  linein = oRead.ReadLine()
  ```

  It moves the current pointer to next line of code and reads next line which is passed to LA.

- **LEXICAL ANALYSER:** The lexical analyser enjoys a key role in this phase of the project. The pieces and lines of code are analysed against the standard syntax of C program. After that the tokens are isolated. For example, if a variable is encountered in the declaration and initialization part then the variable details are being passed on to the variable tracker which takes note of that so that its value and actions on it in
the further program can be noted down and passed on wherever necessary. An example is given below to show the working of lexical analyser.

For example if the analyser encounters the following LOC: #include <stdio.h> the lexical analyser will simply pass information that it is a header file include direction. Another example can be: Void main (). Here the LA will simply indicate the start of programs main body.

STACK MAINTAINER: This is that part of the program which helps and will help later versions of the program to track the scope of syntax and variables by matching the closing and opening braces of the structures. For example: When the LA encounters the following code:

1: While (a==9)
2: {
3: Printf("%d ",A);
4: A=A+1;
5: }

When the LA encounters the while LOC, then it will be indicated that a while loop has started, but yet it is not clear whether the while loop has only one line in its body or has multiple hence having opening and closing braces. So, once the next line is read and passed to LA, it calls push("{") function to indicate the body has started. Then the LA keeps consulting the stack for any the unmatched starting brace counter until then the lines of code have been read and interpreted till it finds any matching brace and updates the stack by pop and updates the counter and checks it to decide whether the loop has finished.

PROGRAM VARIABLE TRACKER: This is the main structure of our VB.NET code in which the program variables are created maintained and over written by new values.
Dim i, k, flag, o, temp1, temp2, flag5 As Integer
Dim varname(10), vartype(10) As String
Dim varvalue(10) As Double
Dim varcounter, j, vvalue, temppos, flag1, flag2, tmp1, tmp2 As Integer
Dim v(15), vname As String

varcounter = -1
i = 0
k = 0
flag = 0
flag2 = 0

**Table 3.1 – Variable Tracker**

- **SLICE COMPRESSOR**: This is the combination of push transformation and kill assignment code. So, as discussed earlier that when a piece of code is encountered which is eligible for PT scanning, and then it will be evaluated, read and killed. The example of .NET code for this phase will be as shown in table below. In this case the statement is neither printed back either to amorphous slice output area and nor printed to the output file as indicated by the comment “‘” mark.

'RichTextBox1.AppendText("varname=" & varname(j) & " varvalue=" & varvalue(j) & vbCr)

**Table 3.2- Slice compression example code**

- **LOC COUNTER**: It is a piece of code which counts the number of lines of code in the source code and slices.

- **VARIABLE COUNTER**: It is a piece of code which counts the total number of variables encountered in the program.
3.2 Tools Used In This Project

Many tools were used in this project which is worth listing below:

- Microsoft Visual Studio -2005
- Microsoft Works / Office – 2003/07
- Ghostscript 8.63
- Adobe Acrobat Reader -8.0
- Microsoft Access -2003/07
- Adobe Photoshop
- Microsoft Paint
- NDF PDF Convertor
- Microsoft MSDN
- Cygwin
- Sprite
- Ponder
- GCC compiler
- Bloodshed Dev C++ compiler
- CIL
- PS to PDF convertor
CHAPTER 4

Data (Program Samples) Testing

Following the construction of various modules of system, this important phase was executed. In this phase some sample programs were shortlisted and passed through this slicing system.

4.1 Tested Programs

Some of the commercially valuable C – Programs were used as a sample to testing and slicing by modeling the real world problems in to a c program structure code. Detailed slides and program code are shown in the appendix of this report [16, 17, and 18]

- Calculator Program to calculate the Visa eligibility points for Tier 1 Visa to UK
- Fibonacci Series Calculator Program

4.2 Comparison of slices Criteria \((v,n)\)

From each program, two variables were selected. One such variable was selected which was commonly assigned, reassigned and used. Other such variable was selected which was seen only few times in the program.

Two main points (line numbers) were chosen for the study, one was at the end of program or at the end of where the major re assignments and usages of variables was done. Other was some where in the middle of the program, at the end of where the declarations were done.

4.3 Limitations of System

Some portion of c program could not be covered. This included some loop structures (except while) of C programs, pointers, structures, unions, functions, strings, characters, goto statements, ++, -- operaors, define structures. [19]
4.4 What Portion of C Was Covered

A version release and test strategy was followed in the project plan. Every version of the system covered some syntax which was more than that covered by its predecessor. To summaries a good portion of c was covered. This included the basic c programs, simple if-else structures, arithmetic operators, declarations, while loops, include files, comments. [19]
CHAPTER 5 - ANALYSIS & CONCLUSION

5.1 Graphs Demonstrating The Studies

A. Graph Of Program 1

![Graph Of Program 1]

B. Graph Of Program 2

![Graph Of Program 2]
5.2 Conclusion & Extended Implication Of Results

If we observe the program 1, we observe that at two points the slices have been taken. One is at line 12 and other is at line 22. At line 12, the program is at the end of declaration stage and at line-22, the program is at the end of calculation stage. It should be noted here that the calculation mainly rotates around the variable points. So, we can see that the point’s variable is being reassigned and reused almost at every line of the calculation part of this program. Keeping in view the above parameters, we can get an idea of the effect of these parameters on the slicing observation and efficiency.

- The syntax slice and amorphous slice at (age, 12) are of same size. The primary reason for this result is that we are obtaining the slice at line-12, where the declaration part ends and the syntax slice filters the unwanted declaration of variables itself in the syntax filtering phase. However, when the slice is passed on to amorphous slicer for further compression, there is nothing to compress as no computation is available except the declarations which the amorphous slicer cannot compress any more.

- Hence, we can conclude that there is no such importance of amorphous slicing as compared to syntax preserved slicing if the point of error is immediately after the declaration part or somewhere around the declaration part.

- The syntax and amorphous slice are again of the same size when a slice of the (points, 12) is obtained. The reasons are two fold. One reason is as stated above; other reason is that even if the variable point has a heavy usage and reassignment at the later stage of program, but at line-12, no effect of that computation is observed as the backward slice is computed at this point.

- Hence, we can conclude that there is no such importance of amorphous slicing as compared to syntax preserved slicing if the point of error is immediately after the declaration part or somewhere around the declaration part even if there is a considerable computation of that particular variable at the later stage of program. This rule applies only to backward slicing algorithms.

- For the slice (age, 22), again the slice size is same for both the algorithms. This is because the variable age is not being reassigned anywhere in between, hence push transformation cannot be applied to the syntax preserved slice. This is true even if the variable age is being used in reassigning the value of variable points.
- Hence, we can conclude that there is no such importance of amorphous slicing as compared to syntax preserved slicing if the point of error is after the calculation part of program and the concerned variable is not being reassigned anywhere in between during the course of program.

- For the slice (points, 22), we can observe the efficiency of amorphous slicing over the syntax preserved slice. It is observed that amorphous slice size is only 59% while as the syntax preserved slice is 95% of the original program. The reason for this difference is twofold. First reason is that as the slice is being computed at the end of computation part. Other reason is that the variable points is being reassigned and reused in many lines of code in the computation part.

- Hence, we can conclude that an amorphous slice can be more efficient if the slice is computed at the end of computation part of program and the slice is being computed with respect to a variable which is heavily used and reassigned in the computation part.

- In program-2, similar conclusions can be made regarding the slicing criteria of (i, 10), (k, 10). However, incase of (I,10) there is a slight difference between amorphous and syntax slice size. This is because of fact that an extra and an unnecessary reassignment is encountered for i. So, if a good programming practice is observed by initializing the correct values at the declaration, ensures that the amorphous slice is not needed at declaration part.

- Hence, we can conclude that a syntax preserved will be equal to amorphous slice in size if a good programming practice is followed by eliminating the redundant reassignments which otherwise could be prevented by initializing the variables by correct vales at the declaration.

- The next very important conclusion is made in this paper by observing the trend of slice sizes while encountering the while loop in program. The difference is observed when slices are computed at the bottom of program. The slice obtained is bigger in size than its amorphous counterpart. The main reason being that a loop squash algorithm was applied.

- Hence, we can conclude that a loop squash algorithm proves to be very much efficient in case of amorphous slices obtained at a point after the loops.
CHAPTER 6

PROJECT PLANNING & PHASES

Although there are many tools to plan a project, but Gantt chart is one of the popular ones and this tool was chosen to plan this project.

6.1 GANTT Chart

What is GANTT Chart

A Gantt chart is a popular type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line. [20]

Advantages and limitations

Gantt charts have become a common technique for representing the phases and activities of a project work breakdown structure (WBS), so they can be understood by a wide audience. A common error made by those who equate Gantt chart design with project design is that they attempt to define the project work breakdown structure at the same time that they define schedule activities. This practice makes it very difficult to follow the 100% Rule. Instead the WBS should be fully defined to follow the 100% Rule, and then the project schedule can be designed. [20]

Although a Gantt chart is easily comprehended for small projects that fit on a single sheet or screen, they can become quite unwieldy for projects with more than about 30 activities. Larger Gantt charts may not be suitable for most computer displays. A related criticism is that Gantt charts communicate relatively little information per unit area of display. That is,
projects are often considerably more complex than can be communicated effectively with a Gantt chart. Gantt charts only represent part of the triple constraints of projects, because they focus primarily on schedule management. Moreover, Gantt charts do not represent the size of a project or the relative size of work elements, therefore the magnitude of a behind-schedule condition is easily miscommunicated. If two projects are the same number of days behind schedule, the larger project has a larger impact on resource utilization, yet the Gantt chart does not represent this difference. Although project management software can show schedule dependencies as lines between activities, displaying a large number of dependencies may result in a cluttered or unreadable chart. [20]

Because the horizontal bars of a Gantt chart have a fixed height, they can misrepresent the time-phased workload (resource requirements) of a project. Activities may appear to be the same size, but in reality they may be orders of magnitude different. A related criticism is that all activities of a Gantt chart show planned workload as constant. In practice, many activities (especially summary elements) have front-loaded or back-loaded work plans, so a Gantt chart with percent-complete shading may actually miscommunicate the true schedule performance status. [20]
Figure 6.1  **Amorphous Slicer Project Plan - Gantt chart**
6.2 **Main Phases Of The Project**

There are mainly two phases of this project. One is a simple syntax preserved slicer which is readily available like Sprite Tool developed already and available under GNU License. ICARIA is a suite of source code analysis tools and libraries for the C programming language including sprite, a program slicing tool for isolating faults in programs by indicating all lines that may have led to a variable having an incorrect value, cgraph, a call graph extractor for showing the relationships between functions, including those called through function pointers, and cawk, a syntactic pattern matcher for displaying or processing all parts of a program matching a given pattern. [21] To the best of my knowledge, only four whole-program slicing tools for C exist: Sprite, CodeSurfer from GrammaTech, Unravel from NIST, and VALSOFT by Jens Krinke. Sprite is a research prototype that is free is use and modify (which many people have done). CodeSurfer is a commercial tool, and Unravel is available for download from NIST. [21] Second phase is the amorphous slicer which basically compresses the syntax preserved slice to the maximum possible extent targeting to achieve a minimum LOC slice. The phase second was built using VB.net technology.

![Figure 6.2-Phases of Project](image-url)
CHAPTER 7

Future Work

Some points need to be mentioned in regard to where this project was left and what are the main aspects from where the project work can be taken further ahead in the field of amorphous slicing.

7.1 Where This Project Was Left

The project execution was done step wise. There were three versions of this project:

1. Version AMSL-1.0 – It covered simple declarations, variable assignments and usages and printings.
2. Version AMSL-1.1 – It covered version 1. In addition to that it covered loop statements. The loop squash algorithm has been included in this version. The loop squash is implemented by including the AMSL-1.0 version – push transformation and kill statement code in a while loop and copying the body of while loop to pseudo lines of codes which are repeatedly called to execute and simulate same number of times as the original while loop permits.
3. Version AMSL-1.2 – It covered version 1 and 2. In addition, work was going on for if-else statements which was left incomplete.

7.2 Future Work

There is a large scope for further future work for this project.

1. Further work on loop statements needs to be done (while has been done).
2. Pointers need to be covered.
3. Structures also need to be covered.
4. Unions are also needed to be covered.
5. Functions also need to be worked upon.
6. Also further work needs to be done in collecting more samples and running them through the system. More random checks need to be done to study the slice sizes of almost all potential variables at various lines of programs.
7. Loop squash algorithm needs to be extended to the user dependent induction variable case of loops.

8. Some further random comparisons have to be conducted in order to make a comprehensive instruction manual as to which type of slicing is to be used in which case in order to attain maximum efficiency and maximum chances of error removal.

9. As we observed in the conclusion that the loop squash algorithm proved to enhance the efficiency of amorphous slices, further algorithms similar to this algorithm need to be developed in other structures of C – programs.
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UML</strong></td>
<td>Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems, referred to as UML model.</td>
</tr>
<tr>
<td><strong>DFD</strong></td>
<td>A data flow diagram (DFD) is a graphical representation of the &quot;flow&quot; of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design).</td>
</tr>
<tr>
<td><strong>Water Fall Development Model</strong></td>
<td>A linear development method where development phases are completed sequentially without overlapping.</td>
</tr>
<tr>
<td><strong>GANTT</strong></td>
<td>A Gantt chart is a popular type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project.</td>
</tr>
<tr>
<td><strong>UML2 Component Diagram</strong></td>
<td>Component-based development (CBD) and object-oriented development go hand-in-hand, and it is generally recognized that object technology is the preferred foundation from which to build components. Some People typically use UML 2 component diagrams as an architecture-level artifact, either to model the business software architecture, the technical software architecture, or more often than not both of these architectural aspects. Physical architecture issues, in particular hardware issues, are better addressed via UML deployment diagrams or network diagrams.</td>
</tr>
</tbody>
</table>
References & Bibliography


APPENDIX
APPENDIX-A - INTERFACE OF THE PROJECT
Some of the views of important pages have been presented below as to indicate the user friendliness and the sophisticated features of the project.
Amorphous Slicing of C-Programs

Mir Abubakr Shahdad                      Department of Computer Science                     King's College London
Amorphous Slicing of C-Programs

Mir Abubakr Shahdad                      Department of Computer Science                     King’s College London
APPENDIX-B- Tested Programs & Slices Obtained

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int age=20;
    salary salary=30;
    int qualification=35;
    int englang=1;
    int funds=1;
    int ukqual=5;
    int points=0;
    printf("Slice point 1 ");
    points=points+age;
    points=points+salary;
    points=points+qualification;
    points=points+ukqual;
    points=points*funds;
    points=points+10;
    points=points*englang;
    points=points+10;
    // slice here
    printf("The Points are = %d",points);
    printf("Slice point 2 ");
}
```

Program 1- Program to calculate the visa eligibility to UK Tier1(general) – PBS

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int age=20;
    printf("Slice point 1 ");
```
Amorphous Slicing of C-Programs

Program1 - Syntax(age,12)

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int age=20;
    printf("Slice point 1 ");
}
```

Program1 - Amorphous(age,12)

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int age=20;
    printf("Slice point 1 ");
}
```

Program1 - Syntax(age,22)

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int age=20;
    printf("Slice point 2 ");
}
```

Program1 - Amorphous(age,22)

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int age=20;
    printf("Slice point 2 ");
}
```
#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int points=0;
    printf("Slice point 1 ");
}

Program1 - Syntax(points,12)

#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int points=0;
    printf("Slice point 1 ");
}

Program1 - Amorphous(points,12)

#include<stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    int age=20;
    int salary=30;
    int qualification=35;
    int englang=1;
    int funds=1;
    int ukqual=5;
    int points=0;

    points=points+age;
    points=points+salary;
    points=points+qualification;
    points=points+ukqual;
    points=points*funds;
    points=points+10;
    points=points*englang;

}
points = points + 10;
printf("The Points are = %d", points);
printf("Slice point 2 ");
}

Program1 - Syntax(points,22)

#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int age=20;
salary salary=30;
    int qualification=35;
    int englang=1;
    int funds=1;
    int ukqual=5;
    int points=0;
    printf("The Points are = 110");
    printf("Slice point 2 ");
}

Program1 - Amorphous(points,22)

#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int age=20;
salary salary=30;
    int qualification=35;
    int englang=1;
    //This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int i=0;
    int j=0;
int k=0;
int n=0;
i=0;
j=1;
printf("Slice point 1");
while(n<=5)
{
k=i+j;
i=j;
j=k;
n=n+1;
}
printf("k value is %d",k);
printf("Slice Point 2");
getch();
}

Program 2- Program to calculate the Fibonacci series of a number N

//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
int i=0;
i=0;
printf("Slice point 1");
}
int i=0;
printf("Slice point 1");
}

//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
int i=0;
int j=0;
int k=0;
int n=0;
i=0;
j=1;
while(n<=5)
{
k=i+j;
i=j;
j=k;
n=n+1;
}
printf("Slice Point 2");
}

a//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
int i=0;
int j=0;
int k=0;
int n=0;
## Amorphous Slicing of C-Programs

**Program2 - Amorphous(i,17)**

```c
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int k=0;
    printf("Slice point 1");
}
```

//This program prints the Fibonacci series

```c
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int k=0;
    printf("Slice point 1");
}
```

**Program2 - Syntax(k,10)**

```c
//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int k=0;
    printf("Slice point 1");
}
```

**Program2 - Amorphous(k,10)**

```c
//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int k=0;
    printf("Slice point 1");
}
```

```c
int i=0;
int j=0;
int k=0;
int n=0;
i=0;
j=1;
while(n<=5)
{
    k=i+j;
i=j;
j=k;
n=n+1;
}
```
Amorphous Slicing of C-Programs

printf("k value is %d",k);
printf("Slice Point 2");
}

Program2 - Syntax(k,17)

//This program prints the Fibonacci series
#include<stdio.h>
#include<conio.h>
void main(void)
{
    int i=0;
    int j=0;
    int k=0;
    int n=0;
    printf("k value is 8");
    printf("Slice Point 2");
}

Program2 - Amorphous(k,17)

APPENDIX C: Source Code of the project

Imports System.Data
Imports System.Data.OleDb

Public Class Form1

    Inherits System.Windows.Forms.Form
    Public con As OleDb.OleDbConnection = New OleDb.OleDbConnection("provider=Microsoft.jet.oledb.4.0;Data source=Database4.mdb;user ID=Admin;Password=")
    Public da As New OleDbDataAdapter
    Public ds As New DataSet
    Public dv As New DataView
    Public path As String = ""

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'MsgBox("hi")
    Label3.Text = path
    da.SelectCommand = New OleDbCommand
    da.SelectCommand.Connection = con
    da.SelectCommand.CommandText = "Select * from paths_temp"

Mir Abubakr Shahdad
Department of Computer Science
King’s College London
da.SelectCommand.CommandType = CommandType.Text
Try
    con.Open()
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

da.Fill(ds, "menu")
dv = New DataView(ds.Tables("menu"))

DataGridView1.DataSource = ds
DataGridView1DataMember = "menu"

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim form2 As New Form2
    form2.Visible = True
    Me.Hide()
End Sub

Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button5.Click
    Dim form2 As New Form3
    form2.Visible = True
    Me.Hide()
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button6.Click
    Application.Exit()
End Sub

Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button8.Click
    If path = "" Then
        MessageBox("Please Make sure A File is selected", MsgBoxStyle.Critical, "Error")
        Exit Sub
    End If
    Form4.Visible = True
    Me.Hide()
End Sub

Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button9.Click
    If path = "" Then
        MessageBox("Please Make sure A File is selected", MsgBoxStyle.Critical, "Error")
        Exit Sub
    End If
    Form4.Visible = True
    Me.Hide()
End Sub
MsgBoxStyle.Critical, "Error")
    Exit Sub

    End If
    Form5.Visible = True
    Me.Hide()
End Sub

Private Sub Label3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label3.Click
End Sub

End Class

Public Class Form2

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        TextBox1.Text = Form1.path
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Application.Exit()
    End Sub

    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label1.Click
    End Sub

    Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim fdlg As OpenFileDialog = New OpenFileDialog()
        fdlg.Title = "Select the Syntax preserved slice C Program"
        fdlg.InitialDirectory = "D:\My Documents\msc_project\samples"
        fdlg.Filter = "All files (*.*)|*.*|C Files (*.c)|*.c"
        fdlg.FilterIndex = 2
        fdlg.RestoreDirectory = True
        If fdlg.ShowDialog() = DialogResult.OK Then
            TextBox1.Text = fdlg.FileName
        End If

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        Dim u As New Form1
        u.Show()
        Me.Close()
    End Sub

End Class
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TextBox1.TextChanged
End Sub

Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
    Form1.path = Me.TextBox1.Text
    Form1.Label3.Text = Me.TextBox1.Text
    Form1.Show()
    Me.Close()
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button6.Click
    Application.Exit()
End Sub

End Class

Public Class Class1

    Public Structure stack

        Shared top As Integer
        'Private arr() As String
        Shared arr() As String
        Private ss As Integer
        Function push(ByVal s As String)
            stack.arr(top) = s
            top = top + 1
        End Function
        Function pop()
            top = top - 1
        End Function
        Function size() As Integer
            Return stack.top
        End Function
        Function isempty() As Boolean

            If top <= 0 Then
                Return True
            Else
                Return False
            End If

        End Function
        End Structure

    End Public

End Class

Form2.vb – Version=ALL
Imports System.Text.RegularExpressions

Public Class Form5
    Private Sub Form5_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim oFile As System.IO.File
        Dim oRead As System.IO.StreamReader
        Dim linein, kk, temp As String
        Dim i, k, flag, o, temp1, temp2, flag5 As Integer
        Dim varname(10), vartype(10) As String
        Dim varvalue(10) As Double
        Dim varcounter, j, vvalue, temppos, flag1, flag2, tmp1, tmp2 As Integer
        Dim v(15), vname As String
        varcounter = -1
        i = 0
        k = 0
        flag = 0
        flag2 = 0
        oRead = oFile.OpenText(Form1.path)
        'oRead.ReadLine()

        While oRead.Peek <> -1
            linein = oRead.ReadLine()
            kk = ""
            For k = 1 To linein.Length
                kk = kk & " ">
            Next
            If kk <> linein Then
                If linein <> "" Then
                    i = i + 1
                    While (linein.StartsWith(" "))
                        linein = linein.Substring(1)
                    End While
                    ' what to skip
                    If linein.StartsWith("#") Or linein.StartsWith("printf") Or linein.StartsWith("//") Or linein.StartsWith("\") Or linein.StartsWith("\") Or String.Compare(linein, ") = 0 Then
                        RichTextBox1.AppendText(linein & vbCr)
                        Continue While
                    End If
                End If
            End If
        End While
    End Sub
'what not to skip

ElseIf linein Like "*main*" Then
    If flag = 0 Then
        RichTextBox1.AppendText(linein & vbCrLf)
        flag = 1
    Else
        MsgBox("Error Multiple Main Declarations",
        MsgBoxStyle.Critical, "Main Error")
    End If

ElseIf flag = 1 Then
    ' Form1.da.SelectCommand.CommandText = "Select * from
paths_temp"
    'RichTextBox1.AppendText(linein & vbCrLf)
    ' check for declarations here
    ' we are checking for ints for some time
    If linein.StartsWith("int") Then
        'RichTextBox1.AppendText(linein & vbCrLf)
        'MsgBox(linein.Split(",").Length)
        For j = 0 To linein.Split(" ","").Length - 1

            varcounter = varcounter + 1
            If j = 0 Then
                v(j) = linein.Split(" ",") (j).Substring(4)
            ElseIf j = linein.Split(" ",").Length - 1 Then
                v(j) = linein.Split(" ",") (j).Trim(";")
            Else
                v(j) = linein.Split(" ",") (j)
            End If

            vname = v(j).Split("=")(0)
            vvalue = v(j).Split("=")(1).Trim(";")
            varname(varcounter) = vname
            varvalue(varcounter) = vvalue
            vartype(varcounter) = "int"

            ' RichTextBox1.AppendText("varname=" &
            varname(j) & " varvalue=" & varvalue(j) & vbCrLf)
        Next

    RichTextBox1.AppendText(linein & vbCrLf)

Else
    ' this part of the program deals with all the code
    without declarations and scan fs and print fs ( mainly re initialisations
    ' and mainly with the computations and re-
    ' let us iterate over the variables in the program
    For o = 0 To varcounter
' check which variables value is being changed

varname(o) = 0 Then
    temppos = o
    Exit For
    Next

'RichTextBox1.AppendText("variable reeinstated " &
varname(o))

'here we check if the variable is being reassigned
by a constant value or by direct value of another variable or by combination of
variables as operands through operators
    ' check for operators on assignment side
    ' let us first remove the end of statement ";
    linein = linein.Replace(";", "")

If linein.StartsWith(".....slice line here") Then
    ' here the slice value is to be computed
    linein = oRead.ReadLine()

    linein = linein.Replace(";", "")

    ' MsgBox("kkk")
    For o = 0 To varcounter
        If String.Compare(linein.Split(",")(1).Trim(" "), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            RichTextBox1.AppendText(linein.Split(",")(0).Remove(linein.Split(",")(0).Length - 3) & " " & varvalue(o) & "\n" & vbCr)
            Exit For
        End If
    Next


    'here we see if some mathematical assignment is being done ( calculation on rhs )

    'now we will check which combination is used

    flag5 = 0

    If linein.Split("=")(1).Contains("+" ) Then
        temp = linein.Split("=")(1)
        For o = 0 To varcounter
            If String.Compare(temp.Split("+")(0),
varname(o)) = 0 Then
    'varvalue(temppos) = varvalue(o)
    flag5 = 1
    temp1 = varvalue(o)

    'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)
    Exit For
End If

Next
If flag5 = 0 Then temp1 =
temp.Split("+")(0)
    flag5 = 0
    ' for variable 2
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        MsgBox(varname(o))
        If String.Compare(temp.Split("+")(1), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            ' temp2 = varvalue(o)
            'RichTextBox1.AppendText(varname(o) & "=" & varvalue(o) & vbCr)
            Exit For
        End If
    Next
If flag5 = 0 Then temp2 =
temp.Split("+")(1)

'ends

' for lhs

temp = linein.Split("=")(0)
For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0 Then
        'varvalue(temppos) = varvalue(o)
        ' temp2 = varvalue(o)
        'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)
        Exit For
    End If
Next
varvalue(o) = temp1 + temp2
' RichTextBox1.AppendText(varname(o) & ";" & vbCr) & varvalue(o) & vbCr
'ends

ElseIf linein.Split("=")(1).Contains("-") Then
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("-")(0), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
        End If
    Next
    If flag5 = 0 Then
        temp1 = temp.Split("-")(0)
        flag5 = 0
        ' for variable 2
        temp = linein.Split("=")(1)
    For o = 0 To varcounter
        ' MsgBox(varname(o))
        If String.Compare(temp.Split("-")(1), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp2 = varvalue(o)
        End If
    Next
    If flag5 = 0 Then
        temp2 = temp.Split("-")(1)
    End If

'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)
Exit For
End If

' for lhs

temp = linein.Split("=")(0)
For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0 Then
        'varvalue(temppos) = varvalue(o)
        ' temp2 = varvalue(o)
'RichTextBox1.AppendText(linein.Split("=")(0) & ";" & vbCrLf) Exit For
   End If

   For o = 0 To varcounter
      If String.Compare(temp.Split("*")(0), varname(o)) = 0
         'varvalue(temppos) = varvalue(o)
         flag5 = 1
         temp1 = varvalue(o)
      End If
   Next

   If flag5 = 0 Then
      temp1 = temp.Split("*")(0)
      flag5 = 0
      'for variable 2
      temp = linein.Split("=")(1)
      For o = 0 To varcounter
         'MsgBox(varname(o))
         If String.Compare(temp.Split("*")(1), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp2 = varvalue(o)
         End If
      Next
      If flag5 = 0 Then temp2 = temp.Split("*")(1)

   'ends
' for lhs

    temp = linein.Split("=")(0)
    For o = 0 To varcounter
        If String.Compare(temp, varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            ' temp2 = varvalue(o)
            'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
            Exit For
        End If
    Next
    varvalue(o) = temp1 * temp2
    ' RichTextBox1.AppendText(varname(o) & "=" & varvalue(o) & vbCrLf)

' ends

' for /

ElseIf linein.Split("=")(1).Contains("/") Then
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("/")(0), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
            'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
            Exit For
        End If
    Next
    If flag5 = 0 Then
        temp1 = temp.Split("/")(0)
        flag5 = 0
        ' for variable 2
    End If
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("/")(1), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
temp2 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & ":" & varvalue(o) & ";" & vbCr)
  Exit For
  End If

Next
  If flag5 = 0 Then temp2 =
  temp.Split("/")(1)

'ends

' for rhs

 temp = linein.Split("=")(0)
 For o = 0 To varcounter
  If String.Compare(temp, varname(o)) = 0
   'varvalue(temppos) = varvalue(o)
   ' temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & ":" & varvalue(o) & ";" & vbCr)
    Exit For
    End If

 Next
  If flag5 = 0 Then temp =
  temp.Split("/")(1)

'ends

End If

Else
  'MsgBox("sss")
  flag1 = 0
  'check if only the other variable is directly being substituted here
  ' so let us iterate again
  For o = 0 To varcounter
    If String.Compare(linein.Split("=")(1), varname(o)) = 0 Then
      varvalue(temppos) = varvalue(o)
      flag1 = 1
      Exit For
    End If
    Next

End If
'If flag1 is not changed at this stage that means a constant assignment is being assigned
If flag1 = 0 Then
  varvalue(temppo s) = linein.Split("=")(1)
End If
End If

End If
End If
End If
End If
End While
Label3.Text = i
Label4.Text = varcounter
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
  Form1.Visible = True
  Me.Close()
End Sub

Private Sub Label3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label3.Click
  End Sub

Private Sub RichTextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RichTextBox1.TextChanged
  End Sub

Public Function IsNumeric(ByVal str As String)
  Dim r As Regex = New Regex("\d+")
  Dim m As Match = r.Match(str)
  If (m.Success) Then
    Return True
  End If
  Return False
End Function

End Class

Form5.vb Version=AMSL-1.0
Imports System.Text.RegularExpressions

Public Class Form5
    Private Sub Form5_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim oFile As System.IO.File
        Dim oRead As System.IO.StreamReader
        Dim linein, kk, temp As String
        Dim i, k, flag, o, temp1, temp2, flag5, whileno, hh As Integer
        Dim varname(10), vartype(10), jj As String
        Dim varvalue(10) As Double
        Dim varcounter, j, vvalue, temppos, temppos1, flag1, flag2, tmp2 As Integer
        Dim v(15), vname, stmtline(10) As String
        varcounter = -1
        i = 0
        k = 0
        flag = 0
        flag2 = 0
        oRead = oFile.OpenText(Form1.path)
        'oRead.ReadLine()

        While oRead.Peek <> -1
            linein = oRead.ReadLine()
            kk = ""
            For k = 1 To linein.Length
                kk = kk & " "
            Next

            If kk <> linein Then

                If linein <> "" Then
                    i = i + 1
                    While (linein.StartsWith(" "))
                        linein = linein.Substring(1)
                    End While

                    ' what to skip
                    If linein.StartsWith("#") Or linein.StartsWith("printf") Or linein.StartsWith("/"") Or linein.StartsWith(""") Or linein.StartsWith("""") Or linein.StartsWith(""") Or String.Compare(linein, "") = 0 Then
                        RichTextBox1.AppendText(linein & vbCrLf)
                        Continue While
                    'what not to skip
                    ElseIf linein Like "*main*" Then
            End If
    End If
    End While
End Sub
End Class
If flag = 0 Then
   RichTextBox1.AppendText(linein & vbCrLf)
   flag = 1
Else
   MsgBox("Error Multiple Main Declarations",
   MsgBoxStyle.Critical, "Main Error")
End If

ElseIf flag = 1 Then
   ' Form1.da.SelectCommand.CommandText = "Select * from

paths_temp"
   'RichTextBox1.AppendText(linein & vbCrLf)
   ' check for declarations here
   ' we are checking for ints for some time
   If linein.StartsWith("int") Then
      'RichTextBox1.AppendText(linein & vbCrLf)
      'MsgBox(linein.Split(",").Length)
      For j = 0 To linein.Split(";").Length - 1
         varcounter = varcounter + 1
         If j = 0 Then
            v(j) = linein.Split(";")(j).Substring(4)
         ElseIf j = linein.Split(";").Length - 1 Then
            v(j) = linein.Split(";")(j).Trim(";")
         Else
            v(j) = linein.Split(";")(j)
         End If
         vname = v(j).Split("=")(0)
         vvalue = v(j).Split("=")(1).Trim(";")
         varname(varcounter) = vname
         varvalue(varcounter) = vvalue
         vartype(varcounter) = "int"
      Next
   RichTextBox1.AppendText(linein & vbCrLf)
Else
   ' while loop squash code starts here
   .....................
   If linein.StartsWith("while") Then
      linein = linein.Split("(")(1)
      linein = linein.Trim(")")
   Else
      ' while loop squash code starts here
      .....................
   End If
jj = linein.Split("<") (0)  
tmp2 = linein.Split("=") (1)  

For o = 0 To varcounter 
  ' check which variables value is being changed 
  If String.Compare(linein.Split("=")(0), varname(o)) = 0 Then 
    temppos1 = o  
    Exit For 
  End If 
Next 

linein = oRead.ReadLine() 

whileno = 0 

' read code of while loop and copy the loc's to an array for repeated calling 

While (Not (linein.Contains(""))) 
  linein = oRead.ReadLine() 
  If Not linein.Contains("")) Then 
    kk = "" 
    For k = 1 To linein.Length 
      kk = kk & " " 
    Next 
    If kk <> linein Then 
      If linein <> "" Then 
        i = i + 1 
        While (linein.StartsWith(" ")) 
          linein = linein.Substring(1) 
        End While 
      End If 
    End If 
  End If 
End While 

stmtline(whileno) = linein 
whileno = whileno + 1 

End If 
End While 

' MsgBox(varvalue(temppos1) & " ----" & tmp2 & vbCr) 
            While (varvalue(temppos1) < tmp2) 
' MsgBox(varvalue(temppos1) & " ----" & 
            tmp2 & vbCr) 
' this part of the program deals with all the code without declarations and scan fs and print fs (mainly re initialisations
' and mainly with the computations and re-
computations
' let us iterate over the variables in the
program

For hh = 0 To whleno - 1
    linein = stmtline(hh)
    For o = 0 To varcounter
        ' check which variables value is being changed
        If String.Compare(linein.Split("=")(0), varname(o)) = 0 Then
            temppos = o
            Exit For
    End If
Next

'ReadTextBox1.AppendText("variable reeinstated " & varname(o))
' here we check if the variable is being reassigned by a constant value or by direct value of another variable or by combination of variables as operands through operators
' check for operators on assignment side
' let us first remove the end of statement ";"
linein = linein.Replace(";", "")

If linein.StartsWith(".....slice line here") Then
    ' here the slice value is to be computed
    linein = oRead.ReadLine()
    linein = linein.Replace(";", "")
    MsgBox("kkk")
    For o = 0 To varcounter
        If String.Compare(linein.Split(",")(1).Trim(""), varname(o)) = 0 Then
            varvalue(temppos) = varvalue(o)
            RichTextBox1.AppendText(linein.Split(",")(0).Remove(linein.Split(",")(0).Length - 3) & " " & varvalue(o) & "";" & vbCr)
        Exit For
    End If
Next

'here we see if some mathematical assignment is being done ( calculation on rhs )

'now we will check which combination is used below

flag5 = 0

If linein.Split("=")(1).Contains("+") Then

temp = linein.Split("=")(1)
For o = 0 To varcounter

If String.Compare(temp.Split("+")(0), varname(o)) = 0 Then

'varvalue(temppos) = varvalue(o)
flag5 = 1
temp1 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)

Exit For

End If

Next

If flag5 = 0 Then temp1 = temp.Split("+")(0)
flag5 = 0

' for variable 2

temp = linein.Split("=")(1)
For o = 0 To varcounter

If String.Compare(temp.Split("+")(1), varname(o)) = 0 Then

'varvalue(temppos) = varvalue(o)

flag5 = 1
temp2 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)

Exit For

End If

Next

If flag5 = 0 Then temp2 = temp.Split("+")(1)

'ends
' for lhs

    temp = linein.Split("=")(0)
    For o = 0 To varcounter

        If String.Compare(temp, varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            ' temp2 = varvalue(o)
            'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
            Exit For
        End If
    End If

    Next

    varvalue(o) = temp1 + temp2

    RichTextBox1.AppendText(varname(o) & "=" & varvalue(o) & vbCrLf)

    'ends

ElseIf linein.Split("=")(1).Contains("-") Then

    temp = linein.Split("=")(1)
    For o = 0 To varcounter

        If String.Compare(temp.Split("-")(0), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
            'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
            Exit For
        End If
    End If

    Next

    If flag5 = 0 Then temp1 = temp.Split("-")(0)

    flag5 = 0
    ' for variable 2

    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        ' MsgBox(varname(o))
        If String.Compare(temp.Split("-")(1), varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
flag5 = 1
temp2 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
  Exit For
End If

Next
If flag5 = 0 Then temp2 =
temp.Split("-"){1}

'ends

' for lhs
temp = linein.Split("=")(0)
For o = 0 To varcounter
  If String.Compare(temp,
  varname(o)) = 0 Then
    'varvalue(temppos) =
    ' temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
    Exit For
  End If
Next
varvalue(o) = temp1 - temp2

RichTextBox1.AppendText(varname(o) & "=" & varvalue(o) & vbCrLf)

'ends

'for * what to do
ElseIf
linein.Split("=")(1).Contains("*") Then
  temp = linein.Split("=")(1)
  For o = 0 To varcounter
    If String.Compare(temp.Split("*")(0), varname(o)) = 0 Then
      'varvalue(temppos) =
      flag5 = 1
      temp1 = varvalue(o)
    End If
  Next
  'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
  Exit For
End If
Next
If flag5 = 0 Then temp1 =

flag5 = 0
' for variable 2

temp = linein.Split("=")(1)

For o = 0 To varcounter
  ' MsgBox(varname(o))
  If String.Compare(temp.Split("=")(1), varname(o)) = 0 Then
    'varvalue(temppos) =
    flag5 = 1
    temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & ":" & varvalue(o) & ";" & vbCrLf)
    Exit For
  End If
Next
If flag5 = 0 Then temp2 =

temp.Split("=")(1)

'ends

' for lhs

temp = linein.Split("=")(0)
For o = 0 To varcounter
  If String.Compare(temp, varname(o)) = 0 Then
    'varvalue(temppos) =
    ' temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & ":" & varvalue(o) & ";" & vbCrLf)
    Exit For
  End If
Next
varvalue(o) = temp1 * temp2

RichTextBox1.AppendText(varname(o) & ":" & varvalue(o) & vbCrLf)

'ends
' for /
ElseIf
linein.Split("=")(1).Contains("/") Then
  temp = linein.Split("=")(1)
  For o = 0 To varcounter
    If String.Compare(temp.Split("/")(0), varname(o)) = 0 Then
      'varvalue(temppos) =
      flag5 = 1
      temp1 = varvalue(o)
      'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)
      Exit For
    End If
  Next
  If flag5 = 0 Then temp1 =
  temp.Split("/")(0)
  flag5 = 0
  ' for variable 2
  temp = linein.Split("=")(1)
  For o = 0 To varcounter
    ' MsgBox(varname(o))
    If String.Compare(temp.Split("/")(1), varname(o)) = 0 Then
      'varvalue(temppos) =
      flag5 = 1
      temp2 = varvalue(o)
      'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCr)
      Exit For
    End If
  Next
  If flag5 = 0 Then temp2 =
  temp.Split("/")(1)

'ends

' for lhs
  temp = linein.Split("=")(0)
  For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0 Then
      'varvalue(temppos) =
' temp2 = varvalue(o)

RichTextBox1.AppendText(linein.Split("=")(0) & "+" & varvalue(o) & ";" & vbCr)
    Exit For
End If

Next
varvalue(o) = temp1 / temp2

RichTextBox1.AppendText(varname(o) & "+" & varvalue(o) & vbCr)
    'ends

End If

Else
    'MsgBox("sss")
    flag1 = 0
    'check if only the other variable is directly being substituted here
    ' so let us iterate again
    For o = 0 To varcounter
        If String.Compare(linein.Split("=")(1), varname(o)) = 0 Then
            varvalue(temppos) = varvalue(o)
            flag1 = 1
            Exit For
        End If
    Next
    'if flag1 is not changed at this stage that means a constant assignment is being assigned
    If flag1 = 0 Then
        varvalue(temppos) = linein.Split("=")(1)

    End If
End If

Next
End While
Continue While
End If
' loop squash code for while ends here

' this part of the program deals with all the code
without declarations and scan fs and print fs (mainly re initialisations
' and mainly with the computations and re-
computations

' let us iterate over the variables in the program

For o = 0 To varcounter
' check which variables value is being changed

If String.Compare(linein.Split("=")(0),
varname(o)) = 0 Then
    temppos = o
    Exit For
End If
Next

'RichTextBox1.AppendText("variable reinstalled " &
varname(o))

' here we check if the variable is being reassigned
by a constant value or by direct value of another variable or by combination of
variables as operands through operators

' check for operators on assignment side
' let us first remove the end of statement ";
linein = linein.Replace(";", "")

If linein.StartsWith(".....slice line here") Then
    ' here the slice value is to be computed
    linein = oRead.ReadLine()

    linein = linein.Replace(";", "")
    ' MsgBox("kkk")

    For o = 0 To varcounter
        If String.Compare(linein.Split(",")(1).Trim(""), varname(o)) = 0 Then
            varvalue(temppos) = varvalue(o)
        RichTextBox1.AppendText(linein.Split(",")(0).Remove(linein.Split(",")(0).Length - 3) & " " & varvalue(o) & "") & vbCr
        Exit For
    End If
Next

ElseIf linein.Split("=")(1).Contains("*") Or
linein.Split("=")(1).Contains("+") Or linein.Split("=")(1).Contains("-") Or
linein.Split("=")(1).Contains("") Or linein.Split("=")(1).Contains("\") Then
    ' here we see if some mathematical assignment is
being done (calculation on rhs)
'now we will check which combination is used below

flag5 = 0

If linein.Split("=")(1).Contains("+") Then
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("+")(1), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
        End If
    End For
    If flag5 = 0 Then
        temp1 = temp.Split("+")(0)
        flag5 = 0
    End If
Next

If flag5 = 0 Then temp1 = temp.Split("+")(0)
flag5 = 0
' for variable 2

temp = linein.Split("=")(1)

For o = 0 To varcounter
    'MsgBox(varname(o))
    If String.Compare(temp.Split("+")(1), varname(o)) = 0
        'varvalue(temppos) = varvalue(o)
        flag5 = 1
        temp2 = varvalue(o)
    End If
Next

If flag5 = 0 Then temp2 = temp.Split("+")(1)

'ends

' for lhs

temp = linein.Split("=")(0)

For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0
        'varvalue(temppos) = varvalue(o)
    End If
Then

'
' temp2 = varvalue(o)

' RichTextBox1.AppendText(linein.Split("=")(0) & ";" & varvalue(o) & ";" & vbCr)
Exit For
End If

Next
varvalue(o) = temp1 + temp2

' RichTextBox1.AppendText(varname(o) & ";" & varvalue(o) & vbCr)

' ends

ElseIf linein.Split("=")(1).Contains("-") Then
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("-")(0),
        varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
            ' RichTextBox1.AppendText(linein.Split("=")(0) & ";" & varvalue(o) & vbCr)
            Exit For
        End If
    Next
    If flag5 = 0 Then temp1 = temp.Split("-")
    flag5 = 0
    ' for variable 2
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        ' MsgBox(varname(o))
        If String.Compare(temp.Split("-")(1),
        varname(o)) = 0 Then
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp2 = varvalue(o)
            ' RichTextBox1.AppendText(linein.Split("=")(0) & ";" & varvalue(o) & vbCr)
            Exit For
        End If
    Next
    If flag5 = 0 Then temp2 = temp.Split("-")

    ' ends
temp = linein.Split("=")(0)
For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0
        'varvalue(temppos) = varvalue(o)
        ' temp2 = varvalue(o)
    Next
    varvalue(o) = temp1 - temp2
    'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
End If

ElseIf linein.Split("=")(1).Contains("*") Then
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("*")(0), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp1 = varvalue(o)
        Next
        If flag5 = 0 Then
temp.Split("*")(0)
    flag5 = 0
    ' for variable 2
    temp = linein.Split("=")(1)
    For o = 0 To varcounter
        If String.Compare(temp.Split("*")(1), varname(o)) = 0
            'varvalue(temppos) = varvalue(o)
            flag5 = 1
            temp2 = varvalue(o)
        Next
        'RichTextBox1.AppendText(linein.Split("=")(0) & "=" & varvalue(o) & ";" & vbCrLf)
Exit For
End If

Next
If flag5 = 0 Then temp2 =
temp.Split("*")(1)

'ends

' for lhs
temp = linein.Split("=")(0)
For o = 0 To varcounter

If String.Compare(temp, varname(o)) = 0

'varvalue(temppos) = varvalue(o)
' temp2 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & ";" & varvalue(o) & "v"
Exit For
End If

Next
varvalue(o) = temp1 * temp2

'RichTextBox1.AppendText(varname(o) & ";"

'ends

' for /
ElseIf linein.Split("=")(1).Contains("/") Then
temp = linein.Split("=")(1)
For o = 0 To varcounter

If String.Compare(temp.Split("/")(0),

'varvalue(temppos) = varvalue(o)
flag5 = 1
temp1 = varvalue(o)

'RichTextBox1.AppendText(linein.Split("=")(0) & ";" & varvalue(o) & "v"
Exit For
End If

Next
If flag5 = 0 Then temp1 =
temp.Split("/")(0)

flag5 = 0
' for variable 2
temp = linein.Split("=")(1)

For o = 0 To varcounter
    ' MsgBox(varname(o))
    If String.Compare(temp.Split("/")(1),
        varname(o) = 0 Then
        'varvalue(temppos) = varvalue(o)
        flag5 = 1
        temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & ";" & vbCr)
    Exit For
End If
Next
If flag5 = 0 Then temp2 = temp.Split("/")(1)

'ends
' for lhs

temp = linein.Split("=")(0)
For o = 0 To varcounter
    If String.Compare(temp, varname(o)) = 0
        'varvalue(temppos) = varvalue(o)
        ' temp2 = varvalue(o)
    'RichTextBox1.AppendText(linein.Split("=")(0) & ";" & vbCr)
    Exit For
End If
Next
varvalue(o) = temp1 / temp2
    ' RichTextBox1.AppendText(varname(o) & ";" & vbCr)

'ends
End If

Else
    'MsgBox("sss")
    flag1 = 0
    'check if only the other variable is directly
    being substituted here
    ' so let us iterate again

    For o = 0 To varcounter
        If String.Compare(linein.Split("=")(1),
            varname(o) = 0 Then
            varvalue(temppos) = varvalue(o)
            flag1 = 1
            Exit For
End If
Next
'if flag1 is not changed at this stage that means a constant assignment is being assigned
If flag1 = 0 Then
  varvalue(temps) = linein.Split("=")(1)
End If
End If

End While
Label3.Text = i
Label4.Text = varcounter
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Form1.Visible = True
Me.Close()
End Sub

Private Sub Label3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label3.Click
End Sub

Private Sub RichTextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RichTextBox1.TextChanged
End Sub

Public Function IsNumeric(ByVal str As String) As Boolean
Dim r As Regex = New Regex("\d+")
Dim m As Match = r.Match(str)
If (m.Success) Then
  Return True
End If
Return False
End Function

End Class