UNIVERSITY OF CALGARY

Can program code be presented in a more aesthetically pleasing fashion?

by

Margaret Frances Nielsen

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

MARCH, 2008

Your file Votre référence
ISBN: 978-0-494-38107-6
Our file Notre référence
ISBN: 978-0-494-38107-6

Abstract

Computer programmers spend hours looking at program code on monitors both during the development and maintenance stages of programming. Basic guidelines for formatting code trace back to the 1970s and have remained virtually unchanged. Research has shown that web-sites and alphanumeric-displays developed to be aesthetically pleasing generate significant positive, and ultimately economic, benefits. Can computer code be manipulated to create a more aesthetically pleasing appearance and thus generating similar positive benefits? This thesis addressed the feasibility of aesthetically enhancing the appearance of computer code through a user preference survey. The computer code was modified using logic-based formatting principles based on Gestalt rules. The survey's main result provided directional evidence that computer code can indeed be reformatted to have a more aesthetically pleasing appearance.

Acknowledgments

I am a mature student with 25+ years working as a reservoir engineer in the oil industry. On retiring, I was fortunate to be able to indulge my fascination with computing science and immerse myself in academia. Gaining my BSc in Computer Science served only to sharpen my urge to gain knowledge.

The inspiration for this work came from an idea that tweaked the imagination of my supervisor, Dr. John Aycock. I appreciate being offered the opportunity to undertake this challenge particularly as this line of research took me far outside my normal technical comfort zone, a true learning experience.

I am grateful to Dr. Aycock who took me on as a student, even knowing that my reasons for seeking an MSc were not those of the "normal" graduate student, and that at my stage in life there would be distractions that would prevent me devoting full-time attention to my studies. He provided support, direction and friendship. He would get me back on track after I had been distracted by other issues in my life. Most of all, I am grateful for his sense of humour, which made it a pleasure to work with him.

I appreciate the contribution of my friends, family, colleagues, and acquaintances who contributed to my work by filling out my questionnaire and acting as never-ending sounding boards. I could not have done it without them.

And finally, I must express my deep gratitude to my husband, Allan, for his moral and financial support in my quest for knowledge -- an expensive hobby to be sure.

Table of Contents

Table of Contents

Approval Page ii

Abstract iii

Acknowledgments iv

Table of Contents v

List of Tables vii

List of Figures and Illustrations viii

Chapter One: Introduction 1
- 1.1 Motivation 1
- 1.2 Summary of the Survey Results 4
- 1.3 Thesis Layout 6

Chapter Two: Aesthetics 8
- 2.1 Aesthetics the Philosophy 8
- 2.2 Aesthetic Affect 11
- 2.3 Chapter Summary 15

Chapter Three: Aesthetic Affect and Computing Science 16
- 3.1 Aesthetics of Computer Hardware 16
- 3.2 Aesthetics of Human/Computer Interaction 18
- 3.3 Aesthetics of Programming 22
- 3.4 Aesthetics of Computer Code 23
- 3.5 Chapter Summary 29

Chapter Four: Aesthetics – Basis for reformatting Code 31
- 4.1 "Good" Web-design Design Practice 31
- 4.2 Typography 35
- 4.3 Overlap of Web-page and Typographical design Rules 39
- 4.4 Chapter Summary 41

Chapter Five: Survey Design and Implementation 42
- 5.1 Objective and rational 42
- 5.2 Survey Plan 42
- 5.3 Selection of Code 45
- 5.4 Code Modifications 50
- 5.5 Survey Implementation 58

5.6 Chapter Summary 59

Chapter Six: Analysis and Survey Results...61
6.1 Data Analysis 61
6.2 Survey Discussion...70
6.3 Conclusion 75
6.4 Chapter Summary 78

Chapter Seven: Thesis Summary and Contibutions..80

References.. 83

Appendix A: Advertisement for Survey Participants 91

Appendix B: Code Snippets 92

Appendix C: Code Comparison Cases 110

Appendix D: Ethics Committee Approval...124

List of Tables

Table 5-1 Elements of User Preference Survey 45
Table 5-2 Code Modifications 46
Table 5-3 Design Principles 51
Table 5-4 Summary of Cases 56
Table 6-1 Survey Response Statistics 62
Table 6-2 Code Preference Based on Indifference Analyses 65
Table 6-3 Code Preference for Indent Variation Analyses 66
Table 6-4 Code Preference by Modification Technique 66
Table 6-5 Code Preference by Snippet 67
Table 6-6 Overall Code Preference (12 cases) 67
Table 6-7 Demographic Stratification 68
Table 6-8 Demographic Cross-Analyses 69
Table 6-9 Analyses based on Demographic Stratification 69
Table 6-10 Code Preference based on Demographic Stratification 70
Table 6-11 Cases where Modified Code Preferred 71
Table 6-12 Analyses for Demographic of Computer Exposure 75
Table B-1 List of Code Snippet Figures in Appendix B 92
Table C-1 Code Comparison Case in Appendix C 110

List of Figures and Illustrations

Figure 2-1 Aesthetic Judgement Framework 14
Figure 3-1 Comparison of 'C' Format Versions 25
Figure 5-1 Code Snippets for the Base Cases 47
Figure 5-2 Rhythm Modification for Snippet 1 54
Figure 5-3 Thumbnail Depiction of Code Snippets 57
Figure 5-4 Example of Survey Comparison Page 58
Figure 6-1 Normalized Responses 63
Figure 6-2 Response with Indifference Range 64
Figure B-1 Advertisement for Survey Participants 91
Figure B-1 Base 1 - Snippet 1 93
Figure B-2 Base 2 - Snippet 2 94
Figure B-3 Base 3 - Snippet 3 95
Figure B-4 Base 4 - Snippet 4 96
Figure B-5 Indent 1 - Snippet 1 97
Figure B-6 Indent 2 - Snippet 2 98
Figure B-7 Indent 3 - Snippet 3 99
Figure B-8 Indent 4 - Snippet 4 100
Figure B-9 Margin 1 - Snippet 1 101
Figure B-10 Margin 2 - Snippet 2 102
Figure B-11 Margin 3 - Snippet 3 103
Figure B-12 Align 1 - Snippet 1 104
Figure B-13 Align 2 - Snippet 2 105
Figure B-14 Align 3 - Snippet 3 106
Figure B-15 Group 1 - Snippet 1 107
Figure B-16 Group 2 - Snippet 2 108
Figure B-17 Group 3 - Snippet 3 109
Figure C-1 Case 1 Indent - Snippet 1 111
Figure C-2 Case 2 Indent - Snippet 2 112
Figure C-3 Case 3 Indent - Snippet 3 113

Figure C-4 Case 4 Margin - Snippet 1 114
Figure C-5 Case 5 Margin - Snippet 2 115
Figure C-6 Case 6 Margin - Snippet 3 116
Figure C-7 Case 7 Align - Snippet 1 117
Figure C-8 Case 8 Align - Snippet 2 118
Figure C-9 Case 9 Align - Snippet 3 119
Figure C-10 Case 10 Group - Snippet 1 120
Figure C-11 Case 11 Group - Snippet 2 121
Figure C-12 Case 12 Group - Snippet 3 122
Figure C-13 Case 13 Indent - Snippet 4 123
Figure D-1 Ethics Committee Approval 124

CHAPTER ONE: INTRODUCTION

1.1 Motivation

Computer programmers spend a tremendous amount of time looking at program code on monitors both during the development and maintenance stages of programming. Basic guidelines for formatting code trace back their linage to pioneers like Kernighan and Plauger [1] and have remained virtually unchanged. "Pretty-printing" programs have been developed to generate more aesthetically pleasing format for code presented in the paper medium. Programming environments provide the ability to present key words in color but the basic formatting guidelines addressing indentation, blank line usage, line length, and print density remain locked in a time warp.

Can programming code be presented in a more aesthetically pleasing fashion? Is there any benefit to reformatting computer code? To address these questions, I needed to develop a basic understanding of the meaning of "aesthetics". This was accomplished through an extensive investigation into related work covering a number of disciplines (philosophy, typography, psychology, graphics design, and marketing), each with their own vocabulary. To distil this information and relate it to computer code was a nontrivial exercise.

Historically web-sites (also viewed on monitors) were developed to be transparent. Published works [2-8] have shown that web-sites and alphanumeric displays developed to be aesthetically pleasing enhance a user's ability to utilize the site and even encourage the user to spend more time at the site. This work is not solely attempting to make the visual appearance more attractive, rather these researchers are attempting to manipulate the emotional experiences of the viewers. Within the realm of social science, the emotional and physical state of an individual, created through a response to aesthetic stimuli, is called aesthetic affect or mood [9]. Through the manipulation of the aesthetic appearance of web-pages, researchers demonstrated significant positive benefits; these benefits contributed to enhancing the experience of the viewer. High correlations were found between the perception of interface aesthetics and usability [2, 3]. Aesthetically pleasing layouts were demonstrated to have an effect on a student's motivation to learn [4]. Aesthetic redesign of graphical displays helps individuals learn [5] and helps reduce the time to find and extract information [6, 7]. Aesthetically redesigned displays have increased productivity and reduced error rate [8].

Thus the question arises "can aesthetically pleasing code help programmers both during the development and maintenance phases of programming?" Can code be manipulated to create a more positive aesthetic affect (mood) within the programmers? Through personal experience and anecdotal comments from fellow programmers or fellow students, without a doubt poorly formatted code can create negative visceral responses. Poorly formatted code can instil feelings of frustration, aversion, and possibly confusion. Poorly formatted code creates negative experiences. Conversely, can aesthetically enhanced appearance of code create positive experience or aesthetic affect?

If code can indeed be formatted in a more aesthetically pleasing fashion, programmers would be working in a more aesthetically pleasing frame of mind which in turn may provide similar positive benefits as those achieved through developing more aesthetically pleasing web-sites. Higher levels of focus and concentration could lead to faster understanding or comprehension of code that programmers are being asked to maintain. During programming, the act of focusing on formatting may result in fewer typographical coding errors or errors in algorithm design. Enhancing the performance of the programmer would result in reduction of the overall cost of programming and program maintenance. Programming economics might also be enhanced by having products reach the markets at a faster rate. These programming cost savings could translate into savings for the target clientele. The target clientele would also benefit by having faster access to their desired products.

To address this potential programming enhancement opportunity, the first step is to determine whether or not program code can be formatted in a more aesthetically pleasing fashion. The big question is "what constitutes more aesthetically pleasing appearance for code?" As stated earlier, the formatting guidelines for code have remained virtually unchanged since the advent of the monitor. Computer language research endeavours have provided no clear direction for enhancement. Their work has provided only conflicting results [10]. Formatting guidelines continue to remain based on strong personal preferences. To affect reliable, consistent and non-arbitrary aesthetic enhancements, modification of code needs to be accomplished through a logic-based process. The modifications to code format should be accomplished through the application of a set of rules or principles where these rules would be applied rigorously, systematically, and methodically. This logic-based process also should lend itself to scientific hypothesis testing.

To find this logic-based approach, I looked at the work done by web-design researchers to develop aesthetically pleasing web-sites; as well, I looked at the practices within the field of typography. Both these areas of research are discussed in Chapter 4. Each field of study has generated and validated a suite of design principles. The overlap of these two suites of design principles has provided the basis for attempting to assess the feasibility of enhancing the aesthetic appeal of programming code.

The objective of this thesis was to demonstrate, through the application of the design principles, that computer programming code could be reformatted to achieve a more aesthetically pleasing appearance.

1.2 Summary of the Survey Results

More precisely stated, the main object of this thesis research project was to demonstrate that:

> Computer programming code, enhanced through the application of logic-based format modifications, **will** be more aesthetically pleasing than code formatted using standard GNU programming guidelines.

This demonstration took the form of a user preference survey with the survey being administered to a sample of 40 participants. Code formatted in a current standard was compared to code that was deliberately modified to enhance its aesthetic appearance. The aesthetic preference of viewers would validate or invalidate the above hypothesis. The independent variable (variable subject to change) would be the formatting modifications of the code snippets. The dependent variable (the measured variable) is the change in aesthetic appeal as perceived by the survey participants.

Code snippets, formatted using a currently acceptable formatting standard (GNU), were compared to the same snippet reformatted to enhance its aesthetic appeal. The logic-based approaches utilized to achieve these modifications were based on the design principles of balance, rhythm and unity developed within the fields of web-page design and typography. Base code formatting options were selected to minimize interference from some obvious potential dependent variables such as the use of color, font selection, comments, etc. The potential for secondary dependent variable exists due to certain demographic variables: age group, how long participants had been using a computer, weekly computer usage levels, and level of programming exposure. To evaluate the impact of these demographic characteristics on the main hypotheses, each of these variants became a minor hypothesis. Demographic data was collected during the administration of the survey.

This survey provided evidence that the appearance of code could be enhanced through the application of the design principle of balance. Greater horizontal balance was achieved through increasing the indent depth and through including a left margin. The enhancement of aesthetic appeal was not definitely demonstrated through the application of the design principles of either rhythm (alignment) or unity (grouping). In some of the individual cases associated with these design principles, positive responses were obtained. These positive responses provide enough optimism to warrant further investigation.

The samplings of demographic factors indicate that they are an influence to the perceived aesthetic appeal of code. The demographic factors investigated were age, how long a participant a used computer, the number of hours of use per week, and exposure to programming. Even though each of these factors demonstrated different responses between their strata, this demographic analysis only provided directional insights. The main focus of the survey was to investigate the potential of enhancing the aesthetic appeal of code. For this reason the selection of participant was on a first come basis with no attempt to ensure the strata for each of the demographic factors was adequately represented. The survey did however establish that these factors are significant and need to be addressed more thoroughly in any future work.

1.3 Thesis Layout

The subject matter of this thesis was to address the question; “can computer code be formatted in a more aesthetically pleasing fashion?” In order to address this question, I investigate the philosophy (aesthetics) of art and beauty and I investigate the studies of social scientists into aesthetic affect in Chapter 2.

In Chapter 3, I examined the inclusion of aesthetic affect (mood) within the field of computer science. Historically, design and implementation within the field of computer science was founded on the principles of functionality, reliability, usability, and cost efficiency. The inclusion of aesthetic consideration was thought to be a brazen marketing ploy. With advancing knowledge about aesthetic affect and its associated benefits, the incorporation of aesthetic affect benefits are starting to appear in most aspects of computer science. I have provided a brief review of the use of aesthetic affect within various aspects of computer science: hardware, human computer interaction (HCI), programming, and code formatting.

The focus of my thesis research was to investigate whether or not program code could be reformatted such that it would be more aesthetically pleasing to viewer and potentially provide the viewer with a more positive experience (aesthetic affect). To find a logic-based approach for my investigation, I looked at the work done by web-design researchers to develop aesthetically pleasing web-sites; as well, I looked at practices within the field of typography. Chapter 4 detailed my investigation into these two fields of study. This chapter also provides the rationale for the selection of the design principles of balance, rhythm and unity.

The hypothesis of this thesis was evaluated through a user preference survey. Chapter 5 outlines the survey design and implementation. It also addresses the selection of the code snippets and the application of logic-based modification techniques. In Chapter 6 the survey data were analysed, discussed, and conclusions were drawn. Chapter 7 provides the contributions of this thesis and also provides suggestions for further work.

CHAPTER TWO: AESTHETICS

Can aesthetically pleasing code help programmers during both the development and maintenance phases of programming? Before this question can be addressed, we first need to determine if code can be formatted in an aesthetically pleasing fashion. But what does aesthetically pleasing really mean? In this chapter, I address the meaning of aesthetic and aesthetic affect.

2.1 Aesthetics, the Philosophy

The word "aesthetic" is used in diverse contexts in our daily lives. As a noun it refers to the philosophical question of what is beautiful or what is art; as an adjective, it refers to the subjective emotional response of individuals to stimuli. In daily conversation, the word is typically used in the adjective form. We see it used when referring to beautiful objects, or a pleasing view, as well as within the beauty industry (aestheticians).

Aesthetics (or esthetics) has the following definition according to the online Compact Oxford English Dictionary [11]:

aes·thet·ic or **es·thet·ic** (ĕs-thĕtˈĭk)
adj.
1. Concerned with beauty or the appreciation of beauty.
2. Having a pleasant appearance.

n.
1. A set of principles concerned with the nature of beauty, especially in art.
2. The branch of philosophy which deals with question of beauty and artistic taste.

The use of the word 'aesthetics' in daily life generally addresses the inherent subjective qualities of an object or system: does the object or system generate an emotional response in the viewer/appraiser? For example, statements like, "I find the painting aesthetically pleasing", or, "The view is exhilarating", are purely subjective opinions based on emotional responses.

Aesthetics is a branch of philosophy that addresses the concepts of beauty and ugliness [12-15]. According to Noel Carroll [14](page 4), "Philosophy is an academic discipline that analyses concepts that are key to human practices and activities, including those of enquiry, those of idealistic endeavours, and those of pragmatic endeavours". These concepts are fundamental to human life. It is our ability to address these concepts abstractly that separates us from the rest of the animal world [16].

The study of philosophy traces back to the ancient time of Plato [12-15]. Based on when and where the philosophers lived, they approached the analysis of aesthetics from a multitude of philosophical schools such as existentialism, phenomenology, Marxism, deconstructionism and analytical [14](page 10). Currently in the English speaking world, analytic philosophy, also referred to as Anglo-American philosophy, is the primary approach to philosophy [14](page 4).

Analytic philosophy is not a social science. Concepts are not addressed by running experiments. Issues are addressed conceptually through reflection. The philosopher is not interested in establishing what people currently believe but rather in determining how to apply a concept correctly and justifiably [14](page 12). For example, "a philosopher will attempt to identify the necessary conditions of art, that is, he will attempt to identify the features of a work that it must necessarily possess in order to count as an artwork" [14](page 7). On the other hand, "a social scientist will attempt to discover what most people in a given society are likely to consider art" [14](page 12).

In analytic philosophy, the standard approach in analysing concepts is the method of necessary and sufficient conditions. Concepts are considered to be categories. Applying a certain concept to an object is a matter of classifying it as a member of the relevant category. For example, calling an object an "artwork" involves determining that the object meets the criteria or conditions required for membership in that category [14](page 7). The analytical philosopher would validate an object as art based on an argument like the following:

> "X is an artwork if and only if X is produced with the intention that it possess a certain capacity, namely the capacity to afford aesthetic experience" [14](page 162).

Aesthetics as a separate field of philosophy was not formally introduced until the 18th century. The term was introduced by Alexander Baumgartner to represent the philosophical investigation of the science of sensory knowledge, the essence and perception of beauty and ugliness [12]. Since its formal inception there has been no consensus as to what the field of aesthetics should or must focus on. Over time, philosophers like Baumgartner, Hume, Kant and Hegel all have approached the concept of beauty and art from different perspectives [12]. According to James Fisher [15], the current most popular approach to aesthetics divides this field into two philosophical views: 'The idea of Beauty' and 'The Philosophy of Art and Art Criticism'. Both these views of aesthetics deal with aesthetic properties or qualities which are response-dependent; they are dependent upon human perception.

A subset of these response dependent qualities is expressive or anthropomorphic properties (sad, happy, excited, sombre, etc.). A second subset of these qualities is object-oriented perceptions that don't allude to the mental state of the viewer, such as adjectives like monumental, dynamic, balanced, unified, graceful, elegant, disorganized, etc. [14] (page 157).

A major point of reflection within the philosophy of aesthetics is the root source of these aesthetic properties. Do the properties originate from the object in question (objective perspective) or do they reside within the viewer (subjective perspective)? From the objective perspective, the beauty of the object is solely determined by its physical properties. From the subjective perspective, the aesthetic perception of the object largely depends on the perceiving individual's characteristics [14, 17-19].

2.2 Aesthetic Affect

Unlike Philosophers, who use reflection and logical arguments to address the various philosophical facets of art and beauty, social scientists study the effect aesthetic properties have on their subjects. They are interested in the emotional or physical changes experienced by their subjects due to these aesthetic properties, not in proving their very existence. Within the realm of social science, the emotional and physical state of an individual, created by aesthetic stimuli, is termed aesthetic affect. Affect is the scientific term used to describe a subject's displayed mood [9]. Like their philosophical counterparts, social scientists struggle with the question of whether changes in aesthetic affect are due to the physical properties of the object (objective perception) versus the individual's personal aesthetic response to the object (subjective perception).

While some social scientists are focussing on studying the source (objective or subjective) of the aesthetic properties, others believe humans are affected by all objects or systems they experience [14](page 159). These scientists attempt to quantify the magnitude of change to a person's aesthetic affect and also attempt to characterize the influence of demographics, state of mind, and current location. Does a painting displayed in a non-prominent location have the same affect as if it were displayed in a world-class museum? Is there a significant aesthetic effect bonus where aesthetic effects are not expected or is there only a penalty when expected aesthetic effects are not realized? Does an unexpected aesthetic experience produce an effect? How does nationality, age, sex, and socio-economic standing impact aesthetic affect? These and many more questions are analysed by social scientists through hypothesis testing, not through the logical reflection techniques utilized by philosophers. The social scientists are interested in the perceptions of their subjects and empirical analyses of their subjects' responses.

Because visual aesthetic effect is hypothesized to exist in everyday objects and systems, social scientists believe it influences many of our choices. Norman in *The Design of Everyday Things* [20] and *Emotional Design* [19] theorizes on how aesthetic effect influences humans information processing. He characterizes information processing as consisting of three stages, each impacted by aesthetic affect: visceral, behavioral, and reflective. While Norman describes three stages, other social scientists [17, 21, 22] believe information processing has two. The first stage is a rapid uncontrolled emotional response to the stimulus. This is followed by a conscious controlled cognitive process. According to Lindgaard [23], evidence from neurophysiology is converging with evidence from psychology to support the claim that emotion precedes cognition. The decision to like or dislike an incoming stimulus is based on the interpretation of a visceral response. Some researchers believe the immediate effective reaction colours subsequent cognitive processes [24, 25]. They believe it is very difficult to change a person's opinion when it was formed based on an initial visceral impression. These researchers also study whether or not these early effective impressions can be manipulated. In the field of advertising, exploiting this aspect of information processing is the norm. In other design areas, researchers have started to investigate the impact of aesthetic affect and have started to incorporate it into their designs.

Norman [20] initially argued aesthetic appearance or beauty of everyday artefacts was a component of design but felt that it should be secondary to reliability and functionality. He argued that incorporating aesthetics into the design process was gratuitous or even manipulative. In his subsequent work [19], he recanted this view of aesthetics and proposed that aesthetic appeal and emotion need to play a much larger role (one not just focused on marketing) in the design of everyday things. He now believes that how we view and potentially use artefacts could be altered by altering the aesthetic appearance. Also in this subsequent work, Norman [19] argued that the appearance of an object has an emotional (aesthetic) effect on the user and this effect is always present. Whether the affect is positive or negative, it changes how we think and feel.

Hartmann and Sutcliffe [17] have proposed a different characterization of aesthetic judgment but one that is compatible with Norman's theory [20] on information processing. They believe aesthetic judgment is composed of stable elements, elements specific to the current interaction with the system or object, and variable elements. The following simplified diagram (Figure 2-1) shows their proposed framework:

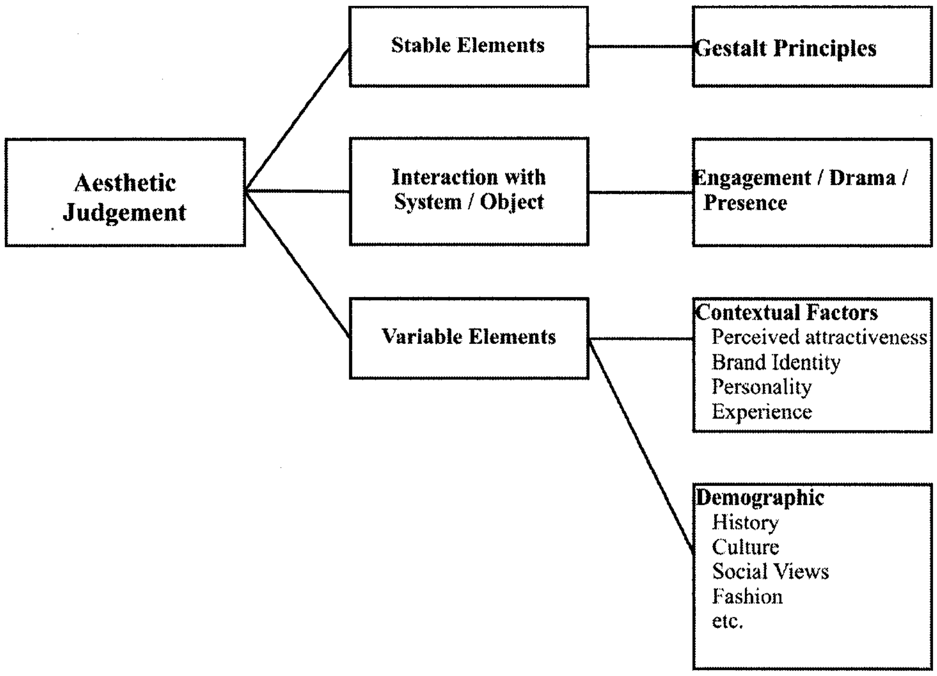


Figure 2-1 Aesthetic Judgement Framework (based on Hartman & Sutcliffe [17])

The stable and variable elements of aesthetic judgment are appreciated through perception, while the interaction elements are appreciated through experience. The variable elements are heavily influenced by demographic factors as well as contextual factors. Contextual factors are generated by the surrounding environment, by the perception of the user, by the current state of mind of the user, by the user's personality and experiences. Demographic influences are based on the user's background, social status, cultural influence, etc. It is relatively difficult to manipulate these demographic influences; however, their effect can be analyzed. The contribution of the interaction elements is based on the generation of pleasure throughout the interaction period. The stable elements can be reasonably measured and are largely independent of demographic factors. These elements can be directly manipulated and the resultant change to aesthetic judgment analyzed. A systematic approach to the manipulation of these stable elements is through the application of the Gestalt rules: symmetry, balance, unity, etc. Gestalt theory is a family of psychological theories that have influenced many research areas since 1924. It is generally accepted that Gestalt theory (expressed as rules) may be used to improve visual design and thereby improve learning [26]. By the appropriate management of aesthetic elements, particularly the stable elements through application of the Gestalt rules, the overall aesthetics generated by the object or systems can be manipulated [26].

2.3 Chapter Summary

Research into the related areas of study, aesthetic and aesthetic affect, provided the necessary foundation to investigate the feasibility of effecting aesthetic modifications to computer code. Aesthetics is a philosophy where issues are addressed through reflection. The aesthetic qualities of an object generate emotional and physical changes in people. These aesthetic responses are studied by social scientists typically through scientific methods, and they have found that visceral responses to aesthetic stimuli precede and influence cognitive thought. Through the manipulation of aesthetic stimuli, aesthetic effect and thus cognitive thought can be influenced. Chapter 3 will specifically examine how aesthetic effect has historically and is currently being addressed in various aspects of computing science design. Chapter 4 will examine how aesthetics have been utilized within the field of web-page design and within the field of typography. The overlap of aesthetic rules from these two fields provides the logic-based approach to attempt to enhance the aesthetic appeal of computer programming code.

CHAPTER THREE: AESTHETIC AFFECT AND COMPUTING AND SCIENCE

The concept of creating aesthetically pleasing impressions or environments is pervasive throughout the field of design [27]. Designers would prefer generating good looking designs over ugly or boring ones. According to Postrel [28] (chapter 2), historically the great majority of people spent their entire income on the basic necessities. As disposable incomes started to increase, spending started to focus on obtaining what everyone else had. Now with disposable income at record levels, spending is focused on obtaining what personally pleases us. Purchase selection is based on look and feel: aesthetics. "Competition has pushed quality so high and prices so low that many manufacturers can no longer distinguish themselves with price and performance, as traditionally defined. In a crowded market place, aesthetics is often the only way to make a product stand out. Quality and price may be absolute but tastes still vary" [28](page 2). However, deliberately harnessing the benefits associated with aesthetic affect has only recently and sparingly been practiced in design [19](page 5)[29, 30].

In this section, the historical and current design practices with respect to aesthetics and aesthetic affect within computing science will be discussed. Points of discussion include hardware design, human/computer interfaces, the design of algorithms, and the formatting of code.

3.1 Aesthetics of Computer Hardware

In the early era of the mainframe computers, the focus of design was to enhance performance of the hardware. The hardware was typically housed in air conditioned clean rooms. These rooms would include card punches, card readers, and tape drives. For programmers, this work environment was noisy and cold. To the average non-computing science person, computer facilities were intimidating. These areas looked like scenes from a science-fiction movie – not anything the average person was likely to sit down at and start trying to operate. The focus of design was to provide functionality and reliability and scarce attention was paid to aesthetics [3, 29, 30].

By 2004, over 72% of Americans used personal computers [31]. The transformation of the computer from a denizen of the science fiction realm to an everyday appliance was due primarily to technological development. Accessing computer resources has moved from punching cards, through the use of monochromatic "dumb" terminals connected to the mainframe, to personal computers connected to the world by the Internet. Traditionally, personal computer design was based on technical features, the cost of manufacturing, and human / system interaction. Marketing ads either stressed the newest high-power features with cost not a major consideration or they featured low price with technical features being secondary. The inclusion of aesthetic appeal was considered to be glitz and irrelevant to the achievement of the goals of reliability and functionality [29].

With the introduction of the Apple's iMac systems, appearance started to be used to market personal computers. The iMac was heralded as the aesthetics revolution in computing [28](page 14). Now appearance / aesthetic appeal is frequently referenced during marketing; for example, flat panel LCD monitors have a "sleek" appearance. In today's market, processing speed and functionality exceed the needs of many users and organizations [32]. As a method to differentiate IT artefacts, the design process is now more oriented towards enhancing the users' experience. Manufacturers are addressing visual appeal and personal preferences.

3.2 Aesthetics of Human/Computer Interaction

Human interaction with objects or systems is studied in the field of ergonomics. As a scientific discipline, ergonomics devotes itself to the study of human-machine systems with the goal of enhancing safety, comfort, productivity and ease-of-use of products and systems. Ergonomics is concerned with the understanding of interactions among humans and other elements of a system. It focuses on design in order to optimize human well-being and overall system performance. Ergonomics is an interdisciplinary study incorporating fields such as anthropology, cognitive science, design, philosophy, psychology, and sociology [33].

Current researchers in the field of ergonomics, such as Liu [34], believe there is an obvious lack of a systematic scientific approach to incorporating aesthetic affect into the ergonomic design of everyday things. Liu argues that aesthetic affects are addressed by industrial and product designers through educated guesses, talent, gut feelings, trends, hunches, or predictions. He also argues that the role of aesthetics and appearance will dramatically increase in the 21st century.

> "To compete and succeed in the market place, manufacturers will have to look beyond reliability and physical quality, and pay more and more attention to the aesthetics and subjective quality of their product" [34] (page 1273).

Liu has proposed that there is a need for more rigorous approaches to incorporating aesthetics into ergonomic design.

While ergonomics is the 'generic' study of human/system interaction, the interaction of users and computers is studied within the field of human computer interfaces (HCI). The goal of HCI is to improve the interaction between the users and the computer systems within both the software and the hardware domains. Research in this field, by universities, corporate, and government labs, has fundamentally changed the face of computing science. An example of these changes is the development of the graphical user interface, whose lineage goes back to research at Xerox PARC. Xerox in turn based their operating system development work on early work at Stanford Research Laboratory and the Massachusetts Institute of Technology [35]. This chain of development greatly altered the usage of computers. Cryptic command line instructions limited computer usage to a select few. The development of the graphical user interface opened the world of computing to the average person.

According to Hewett et al. [36](page 2), HCI addresses many aspects of computing science:

> "Human computer interaction is concerned with the joint performance of tasks by humans and machines; the structure of communication between human and machine; human capabilities to use machines (including the learnability of interfaces); algorithms and programming of the interface itself; engineering concerns arise in designing and building interfaces; … and the process of specification, design, and implementation of interfaces; design trade-offs ."

Traditionally the focus of HCI has been on efficiency, functionality and usability, but researchers (for example [21, 37-39]) have noted that something else is needed beyond these ideals. They suggest a design also needs to incorporate aesthetics: emotions, attraction, and affect. Tractinsky [32](page 1) has provided three arguments as to why aesthetics should be incorporated into HCI design:

> "Aesthetics—plays a major role in our private, social, and business lives. It is argued that aesthetics is relevant to information technology research and practice for three theoretical reasons. (1) For many users, other aspects of the interaction hardly matter anymore. (2) Our evaluations of the environment are primarily visual, and the environment becomes increasingly replete with information technology. (3) Aesthetics satisfies basic human needs, and human needs are increasingly supplied by information technology. Aesthetics matters for a practical reason as well: it is here to stay."

Usability, one of the main tenets of HCI design and evaluation of web-sites, is also in the early stages of being impacted by researchers studying aesthetic affect. Usability researchers have traditionally emphasized performance criteria such as time to learn, error rate and time to complete a task. User experience and satisfaction are traditionally recognized as components of good HCI design; however they are mostly analyzed in terms of their impact on the cognitive information process. Human affect and experience is typically neglected [3, 29, 30]. Tractinsky et al. [40] argues the concept of aesthetic affect and usability represent two orthogonal dimensions of HCI research with the aesthetic affect dimension typically neglected. They also suggest that a gap exists between the practice of much of the computer industry (with current focus on user experience and satisfaction) and the usability/functionality focus of HCI. HCI researchers are ignoring important needs of computer users, who, like consumers of other commodities, are likely to value aesthetics and fashionable designs in addition to usability.

HCI research papers such as [3, 32, 37-42] have argued that aesthetic perceptions of interfaces are highly correlated with perceptions of the interfaces' usability. This correlation suggests that users do not perceive these two design dimensions, aesthetics and usability, as independent. Actual scientific testing has demonstrated that aesthetic affect influences perceived ease-of-use thus colouring the user's overall perception of the website's usability/functionality. Examples of this testing:

- Jordan [43] found that usability and aesthetics are both instrumental in creating pleasurable electronic products.
- Kurosu and Kashimura [2] evaluated the link between the perception of ease of use and the actual usability. They found a high relationship between the user's aesthetic judgment of an interface and its perceived usability.
- Tractinsky [3] and Tractinsky et al. [40] corroborated Kurosu and Kashimura's [2] Japanese findings in a different culture, Israel, thus removing doubt about potential cultural bias.

Although slow, the shift from the traditional focus on cognitive thought process and usability/functionality to addressing the early visceral perception process of aesthetic affect has been noted in studies such as [38, 39]. NORDICHI2005 (summarized by Light [44]) was totally dedicated to the role of aesthetics in HCI. The subsequent conference, NORDICHI2006, also addressed various aspects of aesthetics. The HCI2005 conference in Edinburgh held a workshop on "*Understanding and Designing for Aesthetic Experience*".

Work to investigate this aesthetic affect by researchers like Burmester et al. [37], De Angeli et al. [38, 45], Hassenzahl et al. [46], Krauss [41], Ngo and Burn [47, 48], Ngo et al. [49], Tractinsky [3, 29, 32], Tractinsky et al. [40], Tullis [7, 50, 51] etc. is ongoing. These researchers continue to publish papers and articles that suggest aesthetic affect provides positive benefits far beyond the usual ones achieved by focussing solely on usability/functionality.

3.3 Aesthetics of Programming

In 1959, the members of ACM's Editorial Board (ACM – Association for Computing Machinery) made the following remark as they described the purposes of ACM's periodicals (from [52](page 667)):

> "If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected. … Implicit in these remarks is the notion that there is something undesirable about an area of human activity that is classified as an 'art'; it has to be a Science before it has any real stature."

According to Knuth [52], describing computer programming as a work of art is equivalent to referring to it as an art form. Writing a program can be equated to writing / composing poetry or music. The goal of this art form is to write beautiful programs. A program can provide both intellectual and emotional satisfaction. He also argues that when we read programs authored by others, we can recognize genuine works of art; classifying some as elegant and some as even exquisite. He also argues that it is possible for programmers to write grand programs, noble programs and even truly magnificent ones.

Programming can be equated to literary writing. The manner of expression of a particular writer is produced by choice of words, grammatical structures, and use of literary devices. The combination of all the possible parts of language produces an aesthetic result. Within the realm of programming, the choice of descriptors, formatting, control structures, and algorithm design or choice all are distinctive to a particular programmer. Through the selection and organization of these factors, a programmer creates their own distinctive and possibly aesthetically appealing work.

Computer programming is now considered both a science and an art. It is not a science simply because we refer to the discipline as 'computing science'. Initially programs were written and manipulated until the desired results were obtained. These programs tended to be extremely cryptic and difficult to read due to demands of efficiency and limited memory. With the advancement of technology and with the advancement in our ability to prove a program to be 'correct' [53], the *ad hoc* programming approach evolved to a scientific one. This scientific approach is characterized by words such as logical, systematic, and rational.

Although the science part of computer programming is stressed during programming education, the artistic side still flourishes. This artistic component is characterized by words such as aesthetic, creative, poetic, and elegant. According to Knuth [52], the art and science aspects of computing science nicely complement each other.

3.4 Aesthetics of Computer Code

In the field of computer science the appearance of computer code falls under the subject entitled "style". Style encompasses both the visual layout of the code and its composition. Layout consists of the choice of font and whitespace while composition consists of the choice of constructs, flow of control, and the elegance of the algorithm.

The study of computer programming style can be equated to the study of English literature. English literature has two main components: language (grammar) skills and literary (composition) skills. While in primary school, teaching primarily focuses on the development of language or grammar skills. As the student progresses, the focus switches to the development of literary composition skills. Similarly, in computing science the initial focus in learning how to program is directed toward the understanding of the underlying constructs like assignment and flow control. As the student progresses, the focus switches to the development of algorithms and complexity. Early computer scientists such as Kernighan and Plauger [1] recognized this analogy of program writing to English literature. In their 1974 book *The Elements of Programming Style* [1], they based their FORTRAN guidelines on Strunk and White's English literary guidelines presented in *Elements of Style* [54].

Focus on style was not a consideration in the very early days of computer programming [55]. Due to extreme space and time limitation, efficient and small programs were valued. The result of this myopic but necessary focus was the development of very cryptic programs that were extremely difficult to read and maintain. By the advent of software engineering in the mid 60's, the concept of a computer program having a life-cycle was well accepted. As the maintenance phase accounts for about two-thirds of the total software cost [56](page 69), considerable effort has been directed toward the issue of readability of the source code, that is, style. The advancement of computer architecture drastically reduced space and time constraints and provided the opportunity to achieve considerable readability improvements through the use of layout and simplification of flow control (famously, 'Go To' statements were essentially eliminated or considered harmful [57]).

Every language has numerous resources outlining good programming practice. An underlying theme of good programming practice is the generation of clear, readable source code. These guideline resources are in the form of books, papers and web-pages and address both layout and flow control choices [1, 58-64]. Amazingly, the basic premise of these guidelines has changed very little from those developed in the 70's. The program examples presented in Kernighan's 1978 book [61], *The Elements of Programming Style for C,* are only minimally different from the same examples presented in Horton's 1997 text book [65]. The simple introductory program "Hello World" appeared in both editions of these books. Through examination of the two versions shown below, little has changed.

'C' Programming Language

1978 Version [61]

```
main() {
    printf("hello, world");
}
```

1997 Version [65]

```
int main() {
  printf("hello, world");
  return 0;
}
```

Figure 3-1 Comparison of 'C' Format Versions

Scientific testing was undertaken to look at the effect of style such as including whitespace, indentation levels, flow-chart and the use of comments. Some testing would support the use of style as a means of improving readability [66-69] while other testing suggested these techniques did not significantly improve the readability of source code

[10, 55, 70]. This lack of consensus has been attributed to a number of different theories. Some researchers site the lack of a consistent definition of the programming style concept [10]. Style definitions range from simple approaches addressing only "motherhood and apple pie" concepts (personal opinion based guidelines) [55, 60, 63] to more complex definitions based on the development of in-depth style taxonomies [10].

Testing results are also considered suspect because experiments may have been clouded by intertwining layout and flow control modifications. While testing focused on indentation, flow control was also modified [10]. Similarly, testing with respect to the use of white-space and the use of comments resulted in mixed reviews [10]. The lack of consistent test results was attributed to inconsistent and improper experimental methodology. As well, the test subjects have a wide range of programming skill and programming preferences, and different abilities to adapt to new ideas. Simply seeing new or different code styles subtly effects changes in style. Experimental methodologies have been proposed and tested but still results remain inconclusive.

Although no definitive style has been supported through testing [71], researchers believe good-coding practice is a requirement to achieve readable source code.

Mitchell H. Clifton [64](page 58)	"One of the most important attributes of computer programs is readability. A program that is easy to read and understand is easier to test, maintain, and modify"
Richard J. Miara [68](page 881)	"The consensus in the programming community is that indentation aids program comprehension, although many studies do not back this up."

There are abundant examples of formatting guidelines available, in the form of books, manuals, and web-sites. There are strong similarities between them but each has a slightly different take; the differences are due to personal preferences of the writer.

Accepting as a premise that the prudent use of indentation, white-space, and comments improves the readability of source code, scientists have expended significant research effort into the development of pretty-printers. A pretty-printer takes source code (both formatted and unformatted), applies a set of formatting rules, and generates a new document that is consistently formatted and supposedly more readable.

Pretty-printers range from simple versions that only ensure consistent use of white-space, indentation, and line-breaks to highly sophisticated programs that through the use of font, color, flowcharts, and indexing typically generate the equivalent of published books. Similarly to source code, significant effort has been undertaken to prove that these more consistent and sophisticated documents, generated by pretty-printers, improve the readability of the original source code. Example of this type of research are the works by Baecker [72], Baecker and Marcus [73], and Oman [71].

Software development is typically achieved through team effort and as such it is difficult to maintain coding standards between individuals as well as between distinct teams working on different modules. The use of pretty-printers provides the means of establishing some continuity throughout long, multi-module programs. A major disadvantage is that the resulting pretty-printer product may be very ambiguous and indecipherable at early stages of program development. The choice of package is generally based on a needs assessment, incorporating a large component of personal preference.

Gary T. Leavens [74](page 75) "I have come to view this diversity of pretty-printing styles as a linguistic phenomenon, and thus I can make no (objective) arguments as to why one style should be better than another. Of course, I have my own (subjective) reasons for preferring my favourite style."

Another tool used in the creation of software is the development environment. This tool, like a pretty-printer, provides the means of establishing continuity between program modules and between program team members. However, unlike pretty printers, changes imposed by Development Environments are permanent and very difficult to override.

Both pretty-printers and programming environments use sets of formatting rules to generate their coding products. These rules are for font selection, whitespace usage, indentation levels, and line-break selection. These rules may generate a program consistently formatted and considered more legible but may not generate an aesthetically pleasing visual product.

Martin Ruckert [75](page 39) "In many cases, humans do a much better job at pretty printing--that is making code look pretty--than automated programs do, since true beauty, as in poetry, comes from a structural identity of form and meaning, and it is very difficult to extract the meaning of a piece of code through automated reasoning."

Can an analytic philosophical argument be applied to programming code? (A similar argument, with regard to objects, was presented in Chapter 2.)

> Programming code is considered art if and only if the programmer had intended to create aesthetic properties of beauty and the viewer actually experiences these aesthetic properties.

Do programmers intend to create these aesthetic properties when they generate their code? Do people experience any of the aesthetic properties of beauty when viewing program code? With a positive response to both these questions, perhaps simplistically one could define program code as art. Many programmers spend a considerable amount of time formatting their code to ensure properties of organization, unity, balance, and perhaps even elegance. These same properties can be perceived by readers of the code. Perhaps code developed in this way would pass the philosophical test and could be declared aesthetically appealing, having aesthetic affective qualities. If code does contain or generate aesthetic properties, it may help both programmers and code readers.

3.5 Chapter Summary

This chapter investigated the historical and current influence of aesthetic effect in the area of hardware development, the study of human computer interactions, the design of algorithms and the formatting of code. In hardware development, the focus of design was to provide functionality and reliability and scarce attention was paid to aesthetics. In today's market, processing speed and functionality exceed the needs of many users and as a method to differentiate products, the design process is now more oriented towards enhancing the users' experience. Advancement in code viewing technology (monitors) is astounding; however, this advancement has not influenced the how programming code is formatted.

Research within the field of HCI has fundamentally changed the world of personal computing with the advancement of items like the graphical user interface. Although there have been significant advances the focus of study in this field has traditionally been on efficiency, functionality and usability. There is now a shift to incorporating user experience and satisfaction as important facets of HCI design. These aspects have been shown to provide significant benefits. Chapter 4 investigates how HCI design practices embracing aesthetic appeal could be extrapolated to the programming environment.

Historically the appearance of computer code was addressed under the guise of style. The look and feel of computer code format was and is currently dictated by guidelines based on good programming practice. Research into the benefit of style focused on readability and functionality with scarce emphases on aesthetic appeal. Little change in formatting practice has seen over the last 30 years. The focus of this thesis is to investigate the feasibility of effecting beneficial code modification through enhancing the aesthetic appeal. Chapter 4 looks at how to develop a systematic logic-based methodology to effect aesthetic change. Chapter 5 describes the user preference survey used to determine if code's aesthetic appeal can be enhanced while Chapter 6 provides the analyses of the user survey.

CHAPTER FOUR: AESTHETICS – BASIS FOR REFORMATTING CODE

The focus of my thesis research was to investigate the feasibility of reformatting program code in order to create a more aesthetically pleasing appearance. A survey was administered as a means of addressing this question. Participants were asked to compare two versions of the same code snippet and provide feedback as to their aesthetic preference. The big question is "what constitutes more aesthetically pleasing appearance?" As stated earlier, formatting guidelines for code have remained virtually unchanged since the advent of the monitor. Computer language researchers have provided no clear direction for enhancement; rather this research provided only conflicting results [10]. Formatting guidelines continue to be based on personal preferences. For my study, rather than applying modification based on my personal programming style preferences, aesthetic-oriented code modifications needed to be accomplished through a logic-based systematic approach.

To find a logic-based approach, I looked at the work done by web-design researchers to develop aesthetically pleasing web-sites; as well, I looked at the practices within the field of typography. The overlap of the guidelines from these fields provided the basis for enhancing the aesthetic appeal of programming code.

4.1 "Good" Web Design Practice

Guidelines and empirical studies for creating good screen design started to be published with the advent of computer screens: monochrome and later graphical displays. These personal preference based guidelines advocated "good design" practices that included suggestions to incorporate aesthetics into the design. These aesthetic components were general in nature and not based on theory. According to Ngo and Byrne [47], addressing aesthetics remained in the realm of art rather than science. They contend that due to a lack of formal aesthetic design training, they likely don't possess the appropriate skills required to exact the benefits of aesthetic design.

Web-design research was directed toward issues of acceptability, motivation, learnability, comprehensibility, and productivity. Researchers eventually started to look at enhancing these qualities through the application of aesthetics. Ngo et al. in their 2000 paper [76] cited some of this work:

Acceptability	[2, 3]	High correlations were found between perception of interface aesthetics and usability or acceptability.
Motivation	[4]	Aesthetically pleasing layouts were found to have an effect on a student's motivation to learn.
Learnability	[5]	Aesthetic redesign graphical display helped with the transfer of information.
Comprehensibility	[6, 7]	Aesthetic redesign reduced time to extract information from displays.
Productivity	[8]	Aesthetic redesign reduced processing time and error rate.

Although this work clearly demonstrated a very high correlation between user perception of interface aesthetic and qualities, a problem existed in the research community. According to Reilly and Roach [77], there was an inability to measure or quantify the aesthetic value of an interface. To address this concern, researchers looked back at work addressed to enhancing the appearance of alphanumeric displays.

In 1984 Tullis, as part of his doctoral dissertation [7] (published in subsequent papers [50, 78]), developed and verified a mathematical model that qualified the aesthetics of alphanumeric screen displays. The focus of his study was reformatting alpha-numeric displays in order to enhance readability. He undertook an extensive review of existing literature on computer screen display guidelines. From this literature review, Tullis synthesized measures that characterize alphanumeric displays. Each of these characteristics would generate a specific measurable numeric value. These values were fed into a mathematical model that predicted the mean search time for extracting information as well as predicted a subjective rating of ease of use. These predicted values where then validated though a user study.

Ngo and Byrne [47] focused on developing a verifiable method to quantify the aesthetic appearance of a web-page. Ngo and Byrne synthesized general web-design guidelines and empirical screen layouts data into a mathematical model that assessed the aesthetic appearance of a web-page. A subsequent empirical study demonstrated a close relationship between the perceived subjective opinion of the tested participants and the model generated aesthetic assessment.

Ngo et al. published additional studies in 2000 [76], 2001 [48], and 2003 [49] that reconfirmed that the aesthetic value of web-page display could be quantified. Each of these studies refined their definition of screen aesthetic characteristics as well as provided empirical studies that verified their work. In later papers, the aesthetic characteristics were redefined to mirror Gestalt rules.

In the 1930's and 1940's the Gestalt approach to psychology was developed and became very popular. The seminal speech on this approach was presented by Max

Wertheimer in 1924 [79]. In this approach, psychologists attempt to understand psychological phenomena by viewing these phenomena as organised and structured wholes. They undertook this work through explaining human perception of groups of objects and how we perceive parts of objects. This investigation resulted in the formation of the Gestalt laws of perceptual organization. Depending on how finely these rules are defined, there could be hundreds of rules. Typically these rules are synthesized into a manageable few.

Chang et al. [26] surveyed the Gestalt literature, identifying eleven distinct rules that apply to computer web-page design. These laws were used to redesign an instructional multimedia application. A "user" study found that the new Gestalt based design resulted in overwhelmingly positive results. Viewers found the web-site more aesthetically pleasing, easier to navigate, and demonstrated higher data retention capabilities.

Chang et al.'s [26] Gestalt rules and the design characteristic developed by Ngo and Byrne [47, 48], and Ngo et al. [49, 76] had significant overlap. Parizotto-Ribeiro and Hammond [80] in their work extracted five design principles from this overlap: balance, homogeneity/symmetry, proportion, rhythm, and unity. Parizotto-Ribeiro and Hammond performed a user study to confirm that the application of these five design principles would influence the users' perception of the aesthetics of computer web-pages. A second study by Parizotto-Ribeiro and Hammond [81] confirmed the original results and additionally showed a significant relationship between aesthetics and usability.

Tullis [50, 78], Ngo and Byrne [47, 48], and Ngo et al. [49, 76] developed and verified methodologies to measure the aesthetic value of displays. Tullis [50, 78], Ngo and Byrne [47, 48], Ngo et al. [49, 76], Chang et al. [26], and Parizotto-Ribeiro and Hammond [80, 81] identified and validated (through user studies) a suite of parameters, which if applied, will enhance the aesthetic quality of the web-page or sites. These parameters, Gestalt rules, became the basis for the logic-based aesthetic modifications used in my thesis research. These principles are defined in more detail later.

4.2 Typography

The art of typography in the western world has been practiced for more than 500 years with the advent of this art form in the West associated with the publishing of the Gutenberg Bible in the fifteenth century [82](page 14). Years of trial and error, technical readability studies, and reader preferences studies have resulted in strong guidelines that generate aesthetically appealing printed material: books, magazines, newspapers, etc.

According to Turnbull [82](pages 19-33), printed material is created for the purpose of conveying information. The writer or creator of the information has the advantage over the reader in that they know what they want to say. Writers have at their disposal vocabulary composed of visual elements and have time to compose and reconsider what and how to express their ideas. The reader is at a disadvantage. They must decide on meaning as they read and they can not ask for clarification. Reading is the extraction of information from visual images, words, and pictures. The author creates the message or information but it is the typographer who is responsible for presenting the information in a fashion that allows the reader to efficiently extract and comprehend the intended message.

Authors or writers focus on readability of the printed material. Readability is characterized by the clarity of the writing, the ease of reading, and the human interest in the topic. Readability tests focus on comprehension [82](pages 19-33). Legibility, on the other hand, is the responsibility of the typographer. Legibility stems from how the message appears on the printed material. If all other conditions under which reading is done are constant, the material that is read faster is termed the most legible [82](page 84-85). Legibility tests focus on the form of the written message. Higher legibility is achieved through the application of rules and standards developed in this field over the last 500 years.

Some common technical typographical rules, paraphrased from "*A Psychological Study of Typography*" [83] by Sir C. Burt, "*The Graphics of Communication*" [82] by A. Turnbull and R. Baird and "*Collier's Rules for Desktop Design and Typography*" by D. Collier [84] are:

- Long copy should be broken for easy reading.
- Line length should be 2 ½ alphabets in length - Short lines decrease reading comfort and increase time required for perception. Long lines, particularly in small type impair legibility. Reading is slowed due to the difficulty in picking up the succeeding line after swinging back from the end of the long line.
- Consider increasing leading (space between lines) as line length increases. For ordinary text, one or two points are adequate.
- Don't use too many different faces. Use one face for the body and for emphasis; use a second face for titles.
- Avoid white on black for longer copy. This format is tiring to the eye.
- Use justified left edges - the eye is accustomed to returning to a common point.
- Roman type is preferred – typographers have long contended that legibility is maximized by the use of the standard Roman Faces.
- Stay with 10, 11, and 12 point size for body copy.
- Margins should approach 50% of the page area.

In developing my thesis survey instrument, the above technical typographical rules were applied equally to both the modified and unmodified format versions. The actual application of these rules is discussed in the next chapter.

Legibility is governed by the printed page: face, size, boldness, leading, length of line, margins, even or uneven lines, ink, paper, lighting, and the interest of the reader in the content. Type is more than black marks on paper. These black marks create breaks in the white of the page generating various shapes. The spaces between letters, words, and lines contribute to recognition and interest. When a large number of words are composed, in total they form shapes, textures and tone. When incorporated into a layout, they interact, creating a mood. This aspect of typography is aesthetic appeal.

> "To be observed in the first place, type should be aesthetically pleasing. That is, the face design should be consciously chosen, and letters, words and lines – in fact, the entire composition – should be so displayed as to invite and then sustain attention" [82](page 84).

Within the realm of printed material, each typographer addresses aesthetic appeal through the application of their own personal style. The unique use of general technical guidelines leads to the creation of an individual typographer style. General guiding principles have been developed to provide direction for typographers in their quest to create aesthetically appealing products. According to Turnbull and Baird [82], these general design principles are: balance, contrast, harmony, movement, proportion, rhythm, and unity. These terms have a striking similarity to the design principles presented in the above web-page design subsection. Although not explicitly referenced as such, they are also equivalent in form to Gestalt rules. According to Turnbull and Baird [82](pages 84), psychologists have studied communication via the printed page and their findings have tended to confirm the merit of these guidelines.

These typographical design principles have been incorporated into the design of alphanumeric and graphical computer displays. Computer-display or web-page design researchers like Ozubko [85] (1985), Galizt [86, 87] (1974 and 1981) , Galizt and DiMatteo [88] (1996), Berleant [89] (2000), Harington et al. [90] (2004) have all explicitly drawn from this art form. They have tweaked the typographical rules, principles and guidelines to meet their discipline's requirements.

Baecker [72] used typography principles for pretty printing of "C" source code. He developed a framework for "program visualization" based on principles of effective graphics design. His approach was to enhance the source code legibility through the use of multiple fonts, variable character widths, proportional character spacing, and gray-scale tints. The enhanced source code is output on high-resolution, bit-mapped displays and laser printers. He found a twenty-five-percent increase in the readability of an enhanced source text of C programs as measured by comprehension quiz scores.

Computer code is a form of alphanumeric text. As such, its aesthetic appearance should also be subject to general typographic design rules and technical principles. These validated typographical rules provide an alternative approach to the management of aesthetic appeal. The first approach, based on web-page Gestalt rules, was described in the previous subsection. In the next sub-section, the overlap of these two approaches is discussed.

4.3 Overlap of Web-page and Typographical Design Rules

To evaluate the potential of aesthetically enhancing the appearance of code, a logic-based methodology is required. Both web-page design research and the art of typography have provided the necessary validated rules. Web-design research has provided the rules of balance, homogeneity/symmetry, proportion, rhythm, and unity. The field of typography has provided the general design principles of balance, contrast, harmony, movement, proportion, rhythm, and unity.

The typographical design principles of balance, proportion, rhythm, and unity have direct counterparts within the list of the five web-based Gestalt rules. The typographical principles of contrast, harmony and movement address how print is placed on the page and how print effects eye movement around the page. The web-design rule of homogeneity/symmetry addresses the same issue however it stresses the typographical rule of harmony rather than contrast. Contrast in web-design is utilized to draw one's attention to a particular point or object on the web-page thus altering normal eye movement. In web-design, contrast is considered a tool, used by the web-designer, to achieve artistic flair. It is not considered as a basic rule to enhance the aesthetic value of the web-site. The two different avenues of background research, web-page design and typography, have essentially provided a common set of guidelines that can be used as the basis for attempting to enhance the aesthetic appearance of programming code. These common rules are balance, homogeneity/simplicity, proportion, rhythm, and unity. The definitions of these terms, shown below, are derived from the commonalities of the two fields of study.

Balance – Balance can be defined as the distribution of optical weight in a picture. Symmetrical balance creates a passive space. Asymmetrical balance becomes an active part of the visual presentation. Balance contributes to the sense of order.

Homogeneity / Symmetry – Homogeneity is a measure of how evenly the objects are distributed throughout the screen. Symmetry is the extent to which the screen is symmetrical in three directions: vertical, horizontal, and diagonal. Homogeneity and symmetry create a calmness and ensure (Western cultural) normal eye movement, left to right and top to bottom.

Proportion – The comparative relationship between the dimensions of the screen components. Certain proportions are more attractive to the eye than others.

Rhythm – Rhythm refers to regular patterns of change in elements. It is the orderly repetition of elements, lines, shapes, tones and textures. The eye will spot a rhythm and follow it.

Unity - The extent to which a group of visual elements are perceived as a single entity. Individual elements of the message must be related to each other and to the total design.

In the realm of typography, the use of color is justified only to the extent that it contributes to the realization of three major objectives of graphic communication. Color should be used to attract and retain the attention of the reader. It should enhance the legibility and comprehensibility of the material. Most importantly, it is used to make an impression. According to Turnbull and Baird [82](page 237), the functions of color in printed material are:

- To attract attention – use of contrast
- To produce psychological effects – to create a mood
- To develop associations – link to feelings
- To build retention – color used sparingly can help with retention
- To create and aesthetically pleasing atmosphere – the misuse of color in a message is worse than the use of no color at all

The difficulty with the general use of color is that psychological tests have uncovered personal color preferences based on gender, age, education and geographic locations [82](page 240). Use of color should be left to the experts.

4.4 Chapter Summary

For my study, rather than applying modification based on my personal programming style preferences, aesthetic-oriented code modifications needed to be accomplished through a logic-based systematic approach. To find a logic-based approach, I looked at the work done by web-design researchers to develop aesthetically pleasing web-sites; as well, I looked at the practices within the field of typography; both field provided sets of validated rules. Web-design research has provided the design rules of Balance, homogeneity/symmetry, proportion, rhythm, and unity. The field of typography has provided the general design principles of balance, contrast, harmony, movement, proportion, rhythm, and unity. The overlap of these two sets of rules provided the logic-based approach used to effect aesthetic enhancements to computer code used in a user-preference survey. Chapter 5 describes the survey and actual code modifications. Chapter 6 provides the results, analyses and conclusion of the survey.

CHAPTER FIVE: SURVEY DESIGN AND IMPLEMENTATION

5.1 Objective and Rationale

The objective of this thesis was to demonstrate, through the application of the logic-based design principles, that computer programming code could be reformatted to achieve a more aesthetically pleasing appearance. This demonstration took the form of a user preference survey. Each of three snippets of code was subjected to four logic-based design modifications: 12 comparisons. To address an additional facet of one of the modifications, a 13th comparison was included. Users were asked to compare the unmodified and modified versions and indicate which version they found more aesthetically pleasing. If applicable, they were also allowed to select a no preference option.

The following subsections will address the survey plan, the selection of the code snippets, the logic-based modification techniques, survey implementation, data analysis, and the discussion of survey results and conclusions.

5.2 Survey Plan

Recall that the main object of this thesis research project was to demonstrate that:

> Computer programming code, enhanced through the application of logic-based format modifications, **will** be more aesthetically pleasing than code formatted using standard GNU programming guidelines.

In other words, the population as a whole would prefer the modified code format. This objective was addressed through an experiment, in this case a survey. Information was collected by means of standardized procedures so that every individual was asked the same questions in the same way. The survey's intent was not to describe the particular individuals who are part of the sample but to obtain a composite profile of the population. This survey was administered to a sample of 40 selected participants.

The objective of this survey was to address the above stated research question. Code formatted in a current standard was compared to code that was deliberately modified to enhance its aesthetic appearance. The aesthetic preference of viewers would validate or invalidate the above hypothesis. The independent variable (variable subject to change) was the formatting modifications of the code snippets. Snippets, formatted using a currently acceptable formatting standard (GNU), would be compared to the same snippet reformatted to enhance its aesthetic appeal. The logic-based approaches utilized to achieve these modifications were based on the design principles developed within the fields of web-page design and typography. The actual modifications are discussed in a following subsection. The dependent variable (the measured variable) was the change in aesthetic appeal as perceived by the survey participants. Base code formatting options were selected to minimize interference from some obvious potential dependent variables such as the use of color, font selection, comments, etc. Potential for secondary dependent variable exists due to certain demographic variables: age group, how long participants had been using a computer, weekly computer usage levels, and the level of programming exposure. To evaluate the impact of these demographic characteristics on the main hypotheses, each of these variants became a minor hypothesis. Demographic data was collected during the administration of the survey.

Table 5-1 summarizes the major elements that define this user preference survey:

Hypothesis	Computer programming code, enhanced through the application of logic-based format modifications, **will** be more aesthetically pleasing than code formatted using standard GNU programming guidelines.
Independent Variable	Enhancement to code through formatting – unmodified code based on a GNU standard. Aesthetic enhancements derived from web-design and typographical rules.
Dependent Variable	Perceived aesthetic appeal of the code snippets. Did participants prefer the unmodified versions or modified format versions of the code snippets?
Minor Hypotheses	Aesthetic preferences will not be impacted by age. Aesthetic preferences will not be impacted by the number of years they have been using a computer. Aesthetic preferences will not be impacted by the amount of time they spend each week using a computer. Aesthetic preferences will not be impacted by the level of their programming exposure.
Target Population	The target population was people over the age of 18 whether or not they have had any exposure to computer programming.
Frame	The sample was drawn from adults at the University of Calgary and personal acquaintances. During the Spring 2007 semester, willing participants were solicited through an advertisement (Appendix A). A small monetary reward was offered for participation in the survey. The advertisement targeted Science, Social Science and Fine Arts faculties.
Sample	We targeted a sample size of 25 to 50 adults. Further testing was halted when the sample size reached 40.
Measurement	Survey participants were asked to provide their personal format preference (or indicate no preference) for each of the 13 pairs of code. Each comparison was comprised of an identical snippet of code formatted two different ways: a base formatted using standard GNU guidelines and an aesthetically enhanced version. Code selection and aesthetic enhancements are

	described in a subsequent section.
Instrument	The survey was conducted using an Internet web-site. After logging on to a password protected site, one page of demographic information was gathered. The respondents were then presented 13 comparisons between pairs. The site was set up such that the survey subjects needed to provide a response for each of the comparisons (pages don't advance without a response). The responses from each survey subject were captured in a text file.

Table 5-1 Elements of User Preference Survey

5.3 Selection of Code

There is a profusion of programming languages that could be evaluated in order to see if format modifications could make them more aesthetically pleasing in appearance. However due to time limitation, I only evaluated one programming language. The major degree of freedom in manipulation of code format is the use of white space. Therefore I needed to select a language that is not affected by white-space during compilation or interpretation. Likely candidates are 'C', Java, and Pascal. Since there is an abundance of GNU 'C' code freely available and there are fairly standardized GNU formatting guidelines, I chose GNU 'C'. 'C' is also the language that I am most familiar with.

Although GNU code has fairly standardized formatting guidelines, there is still a fair degree of dissimilarity in appearance between programs; this variance is due to personal style and interpretation of the guidelines. To minimize this style variance, I chose snippets of code from one longer GNU program that I was familiar with: SimpleScalar version 3.0 [91]. This program simulates a modern processor providing the ability to evaluate physical processor modifications. Although programmed by a number of individuals, a fairly stringent set of formatting rules appeared to have been used. The entire program exhibits a similar style.

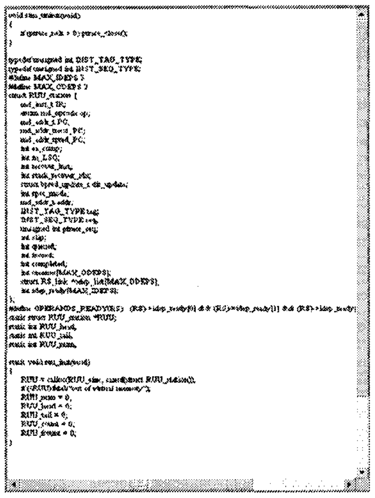
To fairly evaluate formatting modifications, a cross-section of programming structures needed to be incorporated into the study: declarations, assignments, functions, as well as various control structures. From the previous discussion of typography, line length was found to play a significant role in establishing aesthetically pleasing appearance with the preferred line length at about 2.5 alphabets (the 26 letters) [82](page 89). For the purpose of this study, a long line was considered to be greater than 3 alphabets in length.

Rather than attempting to find one snippet of code that contained the entire set of desired attributes, three smaller snippets were chosen. Each snippet contained declarations, functions and assignments. The level of nesting of control structures and the number of long lines varied between the three snippets. A fourth snippet was used to investigate nuances of indentation depth; however, this snippet was not subject to the full suite of modifications. The variations in these two parameters, nesting and line length, for the code snippets are shown below in Table 5-2.

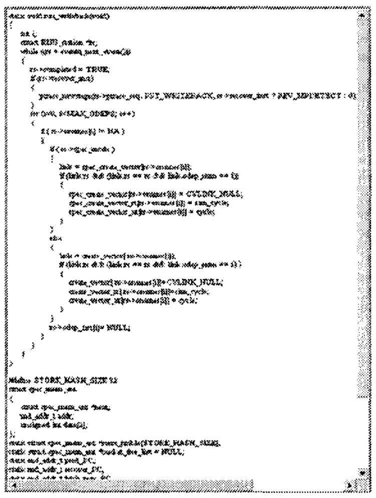
	Level of Nesting	Number of Long Lines
Snippet 1 (Base 1)	1	1
Snippet 2 (Base 2)	7	10
Snippet 3 (Base 3)	3	7
Snippet 4 (Base 4)	7	9

Table 5-2 Code Modifications

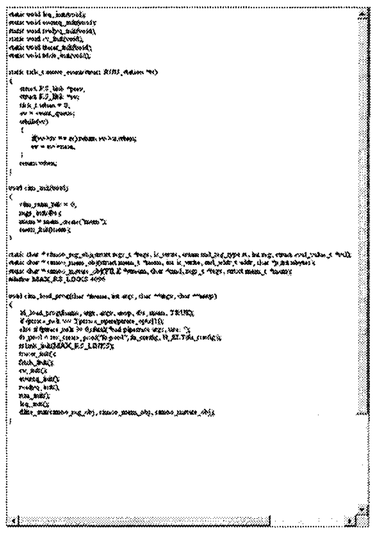
These four snippets which correlate to the respective four base cases are included in Appendix B and are shown below (Figure 5-1) in thumbnail format.



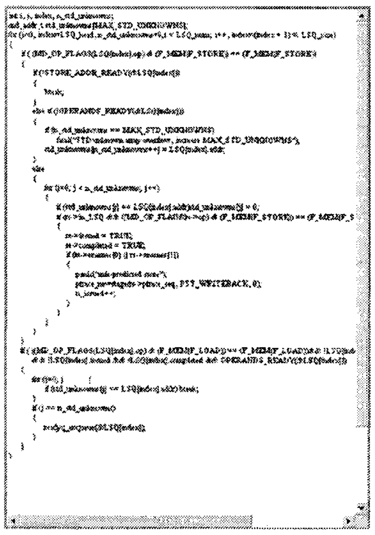
Snippet 1 Minimal Nesting
Minimal Number of Long Lines



Snippet 2 Deep Nesting
Large Number of Long Lines



Snippet 3 Moderate Nesting
Moderate Number of Long Lines



Snippet 4 Deep Nesting
Large Number of Long Lines

Figure 5-1 Code Snippets for the Base Cases

As programmers typically create, read, and modify code on monitors, the survey participants were asked to compare the original code and modified code on a monitor. To simplify the comparison, both the original and modified format versions were shown in individual windows on a 21 inch monitor. To enable both versions to be shown simultaneously, each simulated window was shown at about a 50% normal width (font-size 6pt). This minimized format provided additional benefits. As survey participants were being asked to focus on the appearance of the code snippets, not content, this minimized size discouraged participants' impulses to read and comprehend the code. The minimized format also provided the ability to present longer snippets of code without the need to scroll vertically.

When programmers are asked to critique code, invariably they provide negative feedback on the comments; they find the comments to be insufficient, verbose, unclear, redundant, etc. Providing the correct level of clear comments is very difficult and is strongly subject to personal preference [92]. The inclusion of comments may contribute another dependent variable (measured variable). To remove this potential dependent variable and thus maintain the focus of the participants on the aesthetic appearance of the code, all comments were removed from the code snippets.

Programmers typically work within a programming environment such as VI or EMACS. These environments use color to highlight key words and some constructs. Although there are default color palettes, the programmer can establish a set of colors based on their own personal preferences. Preference testing has shown the color choices are strongly effected by demographics: age, sex, and geographic location [82](page 240). To remove the color as a dependent variable, the snippets were shown with only black fonts on a white background with no emphasis such as bold or italics. Again this was done to maintain the focus of the survey participants on the main dependent variable, aesthetic appearance.

Programming code is similar to a printed page of text. Like printed text, code can be presented in a single column or in a multi-column format. Although the option for multi-columns exists, code is typically viewed as a single column. Also, attempting to compare two columns of code in one window with two columns in a second window would be a more difficult task. For these reasons, the snippets of code were presented in a single column format.

Within the world of typography, an established rule of thumb is that Roman face proportional types should be selected to enhance legibility [82](page 86); therefore, I chose to use Times New Roman proportionally spaced typeface with default kerning (space between letters) for both the unmodified and modified versions of the code snippets. The snippets of code for the base cases are shown in thumbnail format in Figure 5-1. All code cases (modified and unmodified) are shown at full-size (Times New Roman 6pt font) in Appendix B.

5.4 Code Modifications

Modifications were restricted to appearance enhancing techniques that would not affect the fundamental nature of the original source code. For this reason, only formatting options were explored and not techniques such as refactoring or restructuring.

The intent of this thesis study is to demonstrate that the programming code, formatted with current guidelines, can have its appearance aesthetically enhanced through the application of logic-based format modifications. I am not attempting to define guidelines to generate the ultimate overall aesthetically pleasing code, only to demonstrate that there exists an avenue for aesthetic enhancement. If code can be aesthetically enhanced, future study may then address potential benefits.

Rather than using an alternative set of personal preferences to effect format changes, selected modifications were based on the principles developed and validated within the study fields of web-site design and typography. The Gestalt-like design principles associated with both web-design and typography are discussed in Chapter 4. Although developed through two separate fields of study, the independently derived suites of design principles exhibited significant overlap. As discussed in Chapter 4, the overlap of these design principles became the logic basis for aesthetically enhancing the appearance of code. The overlapping design principles developed in Chapter 4 are restated below in Table 5-3:

Balance	Balance can be defined as the distribution of optical weight in a picture. Symmetrical balance creates a passive space. Asymmetrical balance becomes an active part of the visual presentation. Balance contributes to the sense of order.
Homogeneity /Symmetry	Homogeneity is a measure of how evenly the objects are distributed throughout the screen. Symmetry is the extent to which the screen is symmetrical in three directions: vertical, horizontal, and diagonal. Homogeneity and symmetry create a calmness and ensure (Western cultural) normal eye movement, left to right and top to bottom.
Proportion	Proportion is defined as the comparative relationship between the dimensions of the screen components. Certain proportions are more attractive to the eye than others.
Rhythm	Rhythm refers to regular patterns of change in elements. It is the orderly repetition of elements, lines, shapes, tones and textures. The

	eye will spot a rhythm and follow it.
Unity	Unity refers to the extent to which a group of visual elements are perceived as a single entity. Individual elements of the message must be related to each other and to the total design.

Table 5-3 Design Principles

The above design principles provided the overall direction utilized to effect potential aesthetic enhancement to the appearance of the code. The actual technical manipulation of the code entailed manipulation of white space and the minor reordering of lines of code and chunks of code. In an attempt to minimize influences on the main dependent variable (aesthetic appearance), the use of technical reformatting options of font selection, kerning, color, and multiple columns were avoided.

Blatantly altering the sequence of code lines could result in programming fatal faults or erroneous executions. Some minor reordering is available as an option. Reordering independent blocks of code-like functions or re-sequencing lines of declarations does not usually change the execution of the programs. (A full discussion of this topic is outside the scope of this thesis.) Minor reordering was therefore one of the technical options used to achieve change in aesthetic appearance. The other reformatting options involved the manipulation of white space. White space exists within lines, at the beginning of lines (indentation), with the margins, and between words or symbols. The logic-based modifications were achieved through the physical reformatting options listed below.

- Reordering some statements
- Reordering functions
- Repositioning declaration

- Increasing indentation depth
- Adding white space within lines
- Altering margins
- Splitting long lines to create a right margin

These physical reformatting options actually only provided limited scope to achieve potential enhancement in aesthetic appearance. The following examination of the five design principles in association with the physical reformatting options, however, did provide some avenues for investigation.

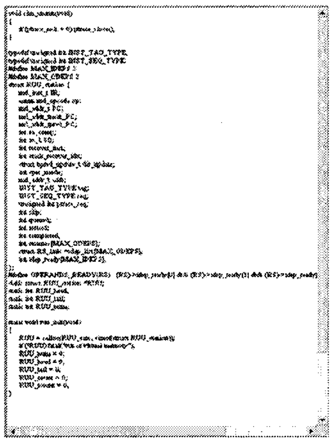
Balance can be defined as the distribution of optical weight with in a picture. Unless deeply nested, the optical weight of computer code is skewed to the left margin. The weight distribution in the vertical direction is dependent on the flow of the program content and thus is not subject to significant modifications. To achieve some degree of horizontal balance, the code could be more centered in the display windows. This can be achieved through two options: increasing the indentation depth and by adding a left margin.

GNU guidelines state that indentation should be 2 to 4 spaces with the actual selection being a personal preference. In the source program SimpleScalar, the indentation depth was set at 4 spaces. In word processing programs such as Microsoft Word and Word Perfect the default indentation, regardless of font size, is a half-inch. With the font selected in the unmodified snippets, a half inch (100% zoom) is achieved through the use of 9 spaces. In the minimal and moderate nested code snippets, the visual weight shifts more to the center line of the display window.

The deeply nested unmodified code snippet presented a challenge. Studies undertaken by researchers addressing the readability of programming code [68, 93] found that deeply nested code became a challenge to read and comprehend when higher levels of indentation depth (greater than 4 spaces) were used. To accommodate the deeply nested lines, the width of the viewing window needs to become excessively wide, lines wrap creating deciphering challenges, or the deeply nested lines need to be split creating very short segments. All of these issues add to the challenge of reading and comprehending. To reduce this excessive shift to the right, a graduated indentation scheme was implemented: 9 spaces, 7 spaces, and everything else at 5 spaces. This accomplished moving the visual weight more to the center of the page without overly exacerbating the long line problem. To evaluate the difference between a constant indentation step and my graduated scheme, an extra comparison (utilizing Snippet 4) was added. In the constant indentation step scheme the indentation depth was held at 9 spaces.

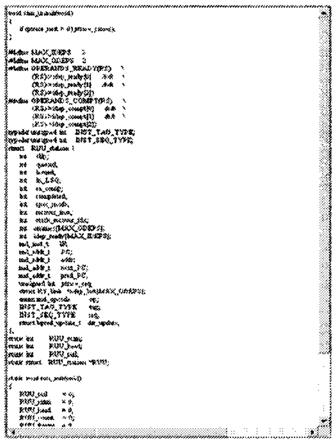
Typically code is presented on a monitor justified completely to the left of the display area. This custom is conjectured to be rooted in the historic display space limitation associated with older small monitors. However pages of printed code, like other printed text, contain both left and right margins. To examine the potential of achieving enhanced visual balance through centering, the inclusion of a left margin was evaluated. In printed mediums, margins range in width. This investigation was not attempting to identify the ultimate margin width; rather it was attempting to investigate the potential aesthetic appearance enhancements. As such, an arbitrary margin width of 15 spaces (equating to about 1 inch) was utilized.

Rhythm refers to regular patterns of change in elements. It is the orderly repetition of elements: lines, shapes, tones and textures. Within the code snippets, some constructs can be grouped and/or reordered, long lines can be split, and inline white space added to create blocks and visual lines. Re-sequencing of constructs or lines was undertaken only if the execution of the program would remain unchanged. The following modifications on code Snippet 1 show the effect of these types of modifications. Code Snippets 2 and 3 were similarly modified. These modifications were complicated and consumed excessive time. The options and choices were many; selecting what to include was a difficult and somewhat arbitrary decision.



```
void clm_startup(void)
{
    if(phase_rock = 0) phase_clone();
}

typedef unsigned int BIST_TAG_TYPE;
typedef unsigned int BIST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_t next_PC;
    md_addr_t pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    BIST_TAG_TYPE tag;
    BIST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   ((RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

Snippet 1: Base Case



```
void clm_startup(void)
{
    if (phase_rock = 0) phase_clone();
}

#define MAX_IDEPS    3
#define MAX_ODEPS    2
#define OPERANDS_READY(RS)      \
        (RS)->idep_ready[0]     &&  \
        (RS)->idep_ready[1]     &&  \
        (RS)->idep_ready[2])
#define OPERANDS_COMP(RS)       \
        (RS)->idep_comp[0]      &&  \
        (RS)->idep_comp[1]      &&  \
        (RS)->idep_comp[2])
typedef unsigned int    BIST_TAG_TYPE;
typedef unsigned int    BIST_SEQ_TYPE;
struct  RUU_station {
    int     slip;
    int     queued;
    int     issued;
    int     in_LSQ;
    int     ea_comp;
    int     completed;
    int     spec_mode;
    int     recover_inst;
    int     stack_recover_idx;
    int     onames[MAX_ODEPS];
    int     idep_ready[MAX_IDEPS];
    md_inst_t     IR;
    md_addr_t     PC;
    md_addr_t     addr;
    md_addr_t     next_PC;
    md_addr_t     pred_PC;
    unsigned int  ptrace_seq;
    struct RS_link  *odep_list[MAX_ODEPS];
    enum md_opcode         op;
    BIST_TAG_TYPE     tag;
    BIST_SEQ_TYPE     seq;
    struct bpred_update_t   dir_update;
};
static int      RUU_num;
static int      RUU_head;
static int      RUU_tail;
static struct   RUU_station *RUU;

static void ruu_init(void)
{
    RUU_tail     = 0;
    RUU_num      = 0;
    RUU_head     = 0;
    RUU_count    = 0;
    RUU_fcount   = 0
```

Snippet 1: Rhythm Modification

Figure 5-2 Rhythm Modification for Snippet 1

Unity refers to the extent to which a group of visual elements are perceived as part of one single piece. Individual elements of the message must be related to each other and to the total design. Code, due to its cryptic looking appearance, provides a certain innate level of overall unity; the entire text is cryptic. Unity could potentially be further enhanced by grouping similar constructs. All global declarations were grouped at the top of each snippet of code. Flow control constructs were made more compact by moving the initializing brace to the same line as the key word. Again long lines were split to create horizontally compact blocks of code.

The web-design/typographical principles of **homogeneity/symmetry** and **proportion** address how visual blocks should be distributed and stressed on the display surface. The necessary vertical sequencing of lines of code (execution) limits modification in these areas. The pretty-printer work by Baecker [72, 92] demonstrated positive benefits could be achieved through creating a zone for comments in the left margin. This allowed the code itself to be displayed uninterrupted. Program code could be displayed utilizing this format but changes to the compiler's parser would be necessary.

One available option under these headings would be to stress key word and identifiers through the use of bold font. The frequency and distribution of these constructs is determined by the content/flow of the program and is not externally controllable for the purpose of enhancing the aesthetic appearance. The potentially haphazard appearance may even reduce the aesthetic appeal of the code. As balance, rhythm and unity had already provided potential aesthetic enhancement options and options available through the application of the principles of homogeneity/symmetry and proportion were more complicated, these two design principles were not utilized.

Appendix B includes a copy of each modification case as well as the base cases. Each modified snippet of code was compared to its unmodified counterpart. Appendix C contains each of these comparisons as the web-page questions presented to survey participants. Survey participants were asked to indicate which option they found more aesthetically pleasing or if they had no preference. The 13 cases are summarized below in Table 5-4 and shown in thumbnail view in Figure 5-3. (Note: this figure does not contain the 13th special indent modification.)

Case	Aesthetic Parameter	Description	Code Snippet
1	Balance	Indent	1
2	Balance	Indent	2
3	Balance	Indent	3
4	Balance	Margins	1
5	Balance	Margins	2
6	Balance	Margins	3
7	Rhythm	Alignment	1
8	Rhythm	Alignment	2
9	Rhythm	Alignment	3
10	Unity	Grouping	1
11	Unity	Grouping	2
12	Unity	Grouping	3
13	Balance	Indentation - constant	4

Table 5-4 Summary of Cases

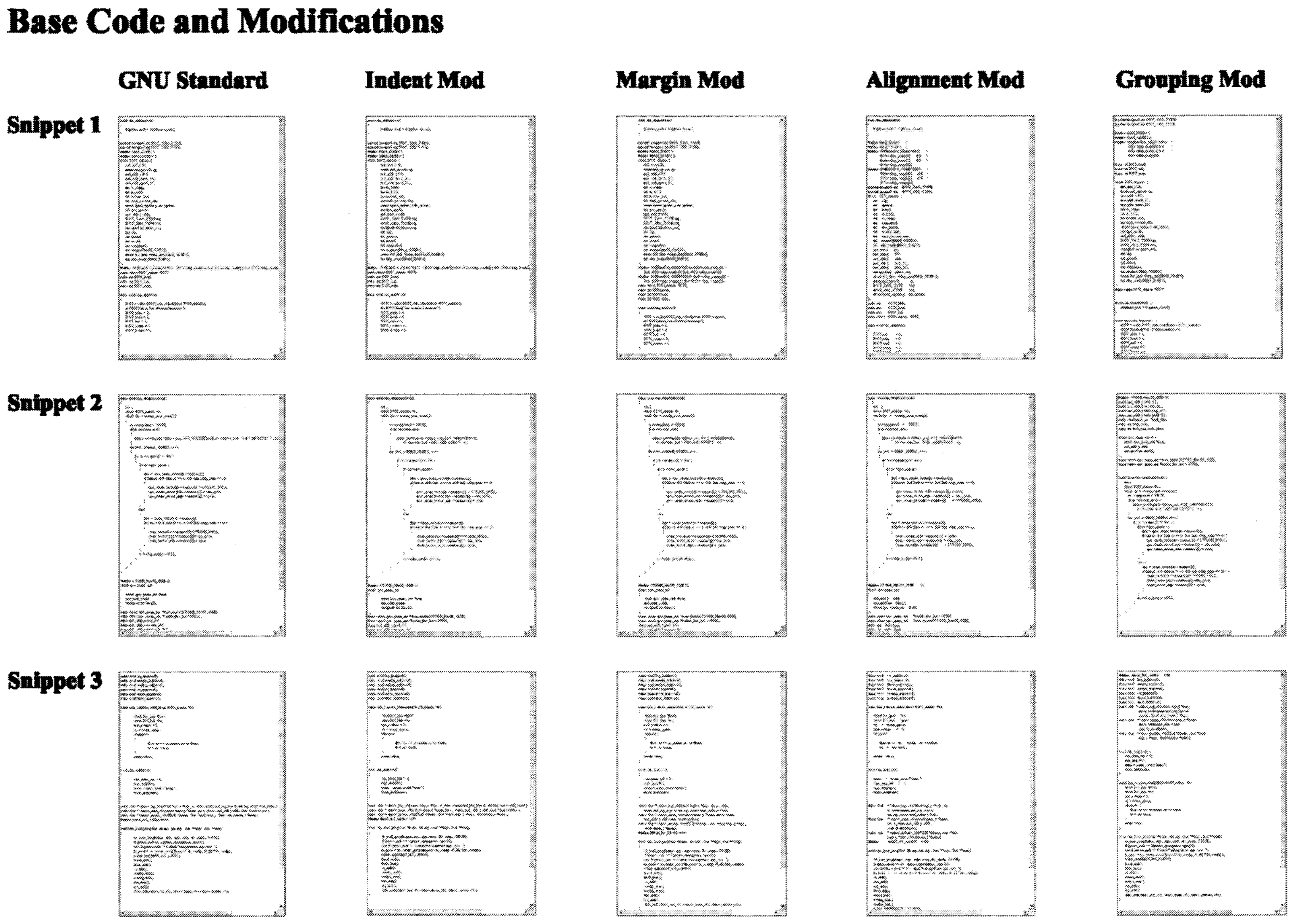


Figure 5-3 Thumbnail Depiction of Code Snippets

5.5 Survey Implementation

The survey instrument was a set of password protected web-pages hosted by the Computer Science department at the University of Calgary. These web-pages consisted of an introductory page, a page to capture demographic information, and thirteen comparison pages. These web-pages can be seen in Appendix C. An example of a comparison page can be seen in Figure 5-4.

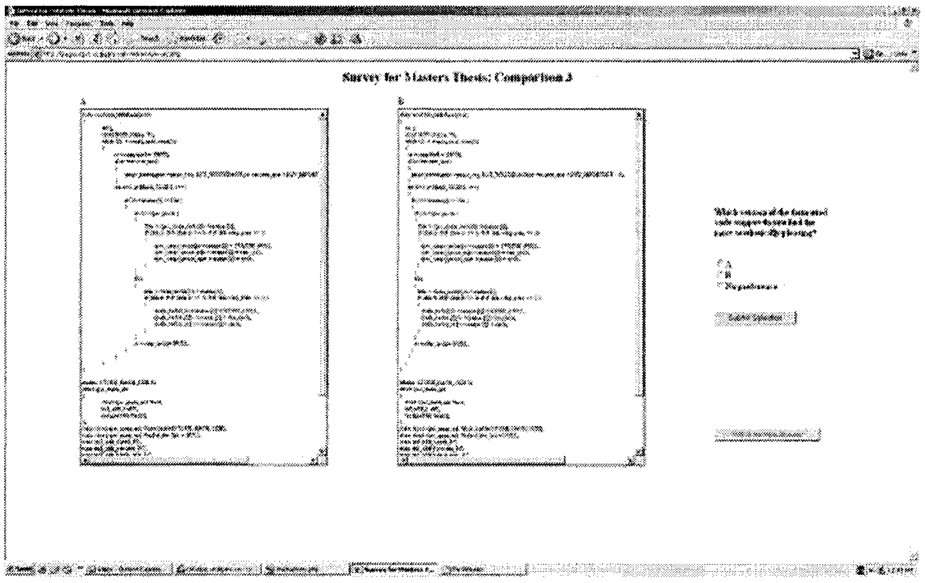


Figure 5-4 Example of Survey Comparison Page

The order of the comparison pages was pseudo-randomized using *Research Randomizer*, an online randomization tool [94]. The placement of the modified and unmodified format versions on either the right or left side of the web-pages was also randomized using *Research Randomizer*. All participants were presented an identical version of the survey, and survey responses were captured in a secure text file also hosted by the Computing Science department.

After minor revisions, required ethics approval from The Conjoint Faculties Research Ethics Board of the University of Calgary was obtained in November 2006. A copy of this approval can be found in Appendix D. Based on the suggestions of the ethics committee, hard-copy participant consent was not required. Rather, reading of the introduction web-page and then proceeding to answer the survey questions was deemed to be consent.

The advertising for survey participants and administration of the survey transpired during the Spring 2007 semester. In an attempt to solicit a broader range of demographic backgrounds, the advertisements were placed primarily in university buildings without a predominance of computing science students or engineering students. A copy of the advertisement can be found in Appendix A. We did not want the sample dominated by individuals with programming exposure. The participants were enticed by a small monetary gift and to affect a random sample, participants were selected on a first-come basis. The survey was administered to all university based participants on the same computer station located in an office away from environmental distractions. Non-university based participants (personal acquaintances) were administered the survey on the system in my home. My home system had a similar configuration to the one at the university and presented a very compatible appearance.

5.6 Chapter Summary

In this chapter, the formal elements of the user preference survey were identified and discussed. The survey was design to address a main hypothesis and a number of minor demographic bases hypotheses. The main hypothesis to be addressed by the survey was:

> Computer programming code, enhanced through the application of logic-based format modifications, **will** be more aesthetically pleasing than code formatted using standard GNU programming guidelines.

The rationalization for programming language selection and the selection of specific code snippets was discussed. The actual modification of the code snippets, based on the design rules developed in Chapter 4, were outlined and justified. This chapter also describes the selection of the 40 survey participants and implementation of the actual survey via the Internet. Chapter 6 provides the analyses and results of the survey.

CHAPTER SIX: ANALYSIS AND SURVEY RESULTS

The survey was administered to 40 participants. Each participant responded to the 12 main comparisons (4 modifications applied to Snippets 1, 2, and 3) and to a 13th comparison undertaken to evaluate potential differences between the two indentation schemes. Each respondent also supplied demographic information. The demographic information was used to evaluate potential additional dependent influences: age, history of computer usage, level of weekly usage, and exposure to programming.

6.1 Data Analysis

For each comparison, the survey participants were given three response options. They could indicate that they preferred the aesthetic appearance of the either unmodified format version, the aesthetic appearance of the modified format version, or they could indicate that they did not have a preference. To evaluate the results, a discrete scale was employed. A nominal value of 1 was applied to a response that preferred the unmodified format version, a value of 2 for a no preference response, and a 3 for a response that preferred the aesthetic appeal of the modified format version. If all 40 participants preferred the aesthetic appearance of the unmodified format version, a single comparison question would generate of score of 40 or a normalized response of 1. If all participants selected the aesthetic appearance of the modified format option, the score would be 120 (normalized response of 3). A middle score of 80 (normalizedresponse of 2) would indicate that the survey participant did not have a preference; they would be indifferent to the formatting options. An indifferent response of 2 is equivalent to the purely random selection of responses for each question.

Statistical information about the population does not exist. Rather the results of this survey were to be used to infer the preference of the population. The population's preference is typically stated as the mean of the sample result with a confidence level of one or two standard deviations. A key underlying assumption in survey analysis is the existence of a normal distribution, but the sample responses did not demonstrate normal distribution for any of the comparison questions. Additionally, each question generated a relatively large standard deviation. The mean ± the standard deviation nearly matched the range of the employed scale. The mean responses and associated standard deviations for the 12 main comparison questions are tabulated below in Table 6-1.

Case	**Aesthetic Parameter**	**Modification**	**Snippet**	**Mean**	**Standard Deviation**
1	Balance	Indent	1	2.45	0.75
2	Balance	Indent	2	1.93	0.94
3	Balance	Indent	3	2.13	0.85
4	Balance	Margins	1	2.10	0.98
5	Balance	Margins	2	2.03	1.0
6	Balance	Margins	3	2.20	0.99
7	Rhythm	Alignment	1	2.13	0.99
8	Rhythm	Alignment	2	1.83	0.81
9	Rhythm	Alignment	3	1.73	0.93
10	Unity	Grouping	1	2.00	0.93
11	Unity	Grouping	2	2.10	1.01
12	Unity	Grouping	3	2.06	0.96

Table 6-1 Survey Response Statistics

The survey confidence level could have been increased by administering the questionnaire to multiple samples. According to the Central Limit Theorem [95], the distribution of the means from multiple samples (even if the individual samples did not demonstrate normal distributions) would approach a normal distribution. The mean of the multiple sample distribution would approximate the preference of the population. Due to cost and time constraints, multiple samples were beyond the scope of this thesis.

For each of these comparisons, we need to determine if the average response could possibly validate the survey's main hypothesis. The normalized response for the 12 comparison questions are plotted on the scale ranging from 1 (total preference for unmodified code) to 3 (total preference for the modified code). Only 10 points show due of duplication of values. Through visual inspection, all responses are clustered toward the middle or indifference area.

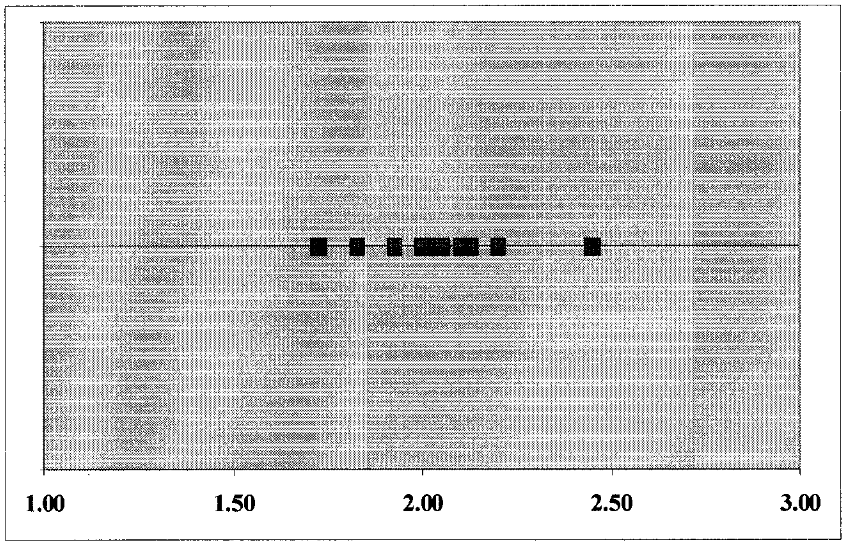


Figure 6-1 Normalized Responses

Total indifference can be equated to the random selection of preference options. By analysing all possible combinations of random selections for a single comparison question (assuming 40 participants), the mean response would be 2 with a standard deviation of 0.035. Figure 6-2 shows the frequency distribution for all possible combinations of random responses to a single comparison question. Superimposed on this figure are the responses for the 12 main questions. The two vertical dashed lines show the range associated with the plus and minus of 2 standard deviations.

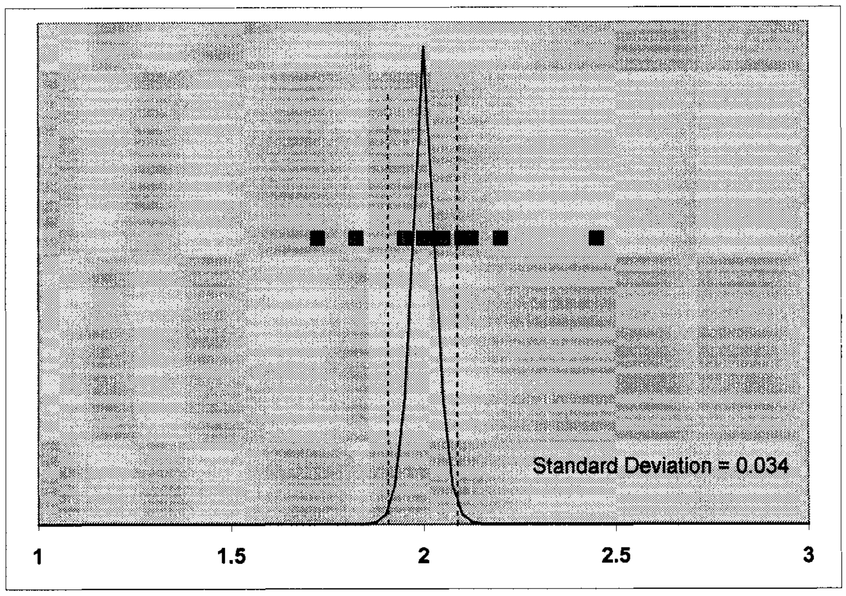


Figure 6-2 Responses with Indifference Range

Four normalized response values fall within the ±2-standard deviation range. These cases, with a 95% confidence level, indicate indifference to the code modifications. The other 8 response values fall outside. These 8 responses directionally indicate that they prefer one of the code versions, either the base case or the modified case.

The four comparisons that demonstrated indifference are cases 2, 5, 10, 12. The other 8 cases show directional preferences for one or the other code formats. The magnitude of that preference can only be inferred by the relative closeness to the ends. When the responses fall to the left of the random range, they can be construed to directionally indicate preference for the unmodified code versions. This group includes cases 8 and 9. Similarly, when the responses fell to the right of the random range, they directionally indicate a preference for the modified version. This group includes cases 1, 3, 4, 6, 7, and 11. Table 6-2 below summarizes this analysis:

Case	Aesthetic Parameter	Modification	Snippet	Average	Code Preference
1	Balance	Indent	1	2.45	Modified
2	Balance	Indent	2	1.93	Indifferent
3	Balance	Indent	3	2.13	Modified
4	Balance	Margins	1	2.10	Modified
5	Balance	Margins	2	2.03	Indifferent
6	Balance	Margins	3	2.20	Modified
7	Rhythm	Alignment	1	2.13	Modified
8	Rhythm	Alignment	2	1.83	Unmodified
9	Rhythm	Alignment	3	1.73	Unmodified
10	Unity	Grouping	1	2.00	Indifferent
11	Unity	Grouping	2	2.10	Modified
12	Unity	Grouping	3	2.05	Indifferent

Table 6-2 Code Preference Based on Indifference Analyses

The 13th code comparison was added to investigate the effect of using a constant indent depth versus the graduated scheme applied in case 2. In the graduated scheme, the indent depths were 9, 7, and then 5 thereafter. In the constant indentation scheme, the depth was held at 9 spaces. The constant scheme was applied to a fourth snippet, highly indented with numerous long lines similar to Snippet 2. The constant indentation scheme excessively shifted the visual center to the right. For this comparison, participants preferred the look of the unmodified code. Table 6-3 compares the results for case 2 and case 13.

Case	**Snippet**	**Average**	**Code Preference**
2	2	1.93	Indifferent
13	4	1.85	Unmodified

Table 6-3 Code Preference for Indent Variation Analyses

This same analysis approach was applied to cases grouped by modification technique, snippet selection, and overall responses to the twelve questions. The results for these analyses are shown in Table 6-4 through Table 6-6.

Aesthetic Parameter (Case)	**Average**	**Code Preference**
Indent (1,2,3)	2.17	Prefer modified code
Margin (4,5,6)	2.11	Prefer modified code
Alignment (7,8,9)	1.89	Prefer unmodified code
Grouping (10,11,12)	2.05	Indifferent

Table 6-4 Code Preference by Modification Technique

Case	Averge	Directional Inference
Snippet 1 (1,4,7,10)	2.17	Prefer modified code
Snippet 2 (2,5,8,11)	1.97	Indifferent
Snippet 3 (3,6,9,12)	2.03	Indifferent

Table 6-5 Code Preference by Snippet

	Average	Directional Inference
Total	2.06	Indifferent

Table 6-6 Overall Code Preference (12 cases)

In the analysis of the survey data shown above, the independent variable was the modifications of the code snippets based on the logic-based principles of balance, rhythm, and unity. The dependent variable was the survey participants' perceived aesthetic appeal. Recall that potential for secondary dependent variables exists due to demographic variables: age group, how long participants had been using a computer, weekly computer usage levels, and the level programming exposure. To evaluate the impact of these demographic parameters, demographic information was collected in the form of strata (range). Survey participants were selected on a first come basis, and no attempt was made to ensure that each of the demographic strata was sufficiently represented. The result was that some of the strata were represented by only one or two sets of responses. For each demographic variable, these under-represented strata were merged, thus each demographic variable evaluated assuming only two strata. Table 6-7 below shows the four demographic categories with the original and merged stratifications. Note that the merging of strata for the programming factor is somewhat skewed. Any potential benefit from creating a more aesthetically pleasing environment would stem from enhancing the performance of programmers. As such, the original demographic strata were merged to represent the two distinct groups; no exposure or non-active programmers and active programmers. In the attempt to prevent the sample from being dominated by individuals with active programming activities, we actually skewed the sample to individuals with no exposure to programming leaving the active programming strata under-represented.

Demographic Factor	Original Stratification	Participants	Evaluated Stratification	Participants
Age	18-29 years old	31	< 30 years old	31
	30-50 years old	6	>= 30 years old	9
	50+ years old	3		
Computer History	never	2	< 6 years	8
	last 2 years	2	>= 6 years	32
	2 to 5 years	4		
	6+ years	32		
Computer Usage	Never / infrequent	1	< 20 hours/week	25
	< 7 hour / week	7	>= 20 hours/week	15
	7 < hours / week < 20	17		
	20 < hours / week	15		
Programming	No programming	24	None or non active	35
	A little in the past	11	Active programming	5
	Program occasionally	2		
	Program frequently	3		

Table 6-7 Demographic Stratification

Each demographic category was cross-analysed with the information from the other categories. The cross-analysed data, shown below, provides strata characterizations. For example, people under 30 years of age tended to have used computers for a longer time than their over 30 counterparts, they used the computer more each week, and had more programming exposure. People who have used their computers for less than 6 years tended to be older and utilized their computer less per week. People who use their computers less than 20 hours per week tend to have used computers for a shorter period time and have had less exposure to programming. People with no programming experience tended to be older and utilized computers less per week than their programming counterparts.

		Demographics			
Categories	**Strata**	**Participants**	**Stratification Description**		
Age	< 30 years	31	Longer History	Higher Weekly hours	More Prog Exposure
	>= 30 years	9	Shorter History	Lower Weekly hours	Less Prog Exposure
History	< 6 years	8	Older	Lower Weekly hours	Similar Prog Exposure
	>= 6 years	32	Younger	Higher Weekly hours	Similar Prog Exposure
Usage	< 20 hours/week	25	Similar Ages	Shorter History	Less Prog Exposure
	>= 20 hours/week	15	Similar Ages	Longer History	More Prog Exposure
Programming	None or Non Active	35	Older	Shorter History	Lower Weekly hours
	Active Programming	4	Younger	Longer History	Higher Weekly hours

Table 6-8 Demographic Cross-Analyses

The responses to the comparison question averaged over the 12 questions and also broken down by modification technique are shown below. Table 6-9 and Table 6-10 show the results of the preference analysis (similar analysis as applied above).

Demographic Factors	**Demographic Strata**	**Normalized Response 12 Comparisons**	**Normalized Response for Modification Technique**			
			Indent	**Margin**	**Align**	**Grouping**
Age	< 30 years	2.01	2.14	2.03	1.83	2.03
	>= 30 years	2.21	2.26	2.37	2.11	2.11
History	< 6 years	2.09	2.29	2.08	2.08	1.92
	>= 6 years	2.04	2.14	2.11	1.84	2.08
Usage	< 20 hours/week	2.05	2.15	2.03	1.93	2.11
	>= 20 hours/week	2.06	2.20	2.24	1.82	1.96
Programming	None or Non Active	2.06	2.16	2.11	1.89	2.08
	Active Programming	2.00	2.20	2.07	1.93	1.80
Overall		2.06	2.17	2.11	1.89	2.05

Table 6-9 Analyses based on Demographic Stratification

Demographic Factors	Demographic Strata	Interpretation 12 Comparisons	Interpretation per Modification Technique Indent	Margin	Align	Grouping
Age	< 30 years	Indifferent	Modified	Indifferent	Unmodified	Indifferent
	>= 30 years	Modified	Modified	Modified	Modified	Modified
History	< 6 years	Modified	Modified	Modified	Modified	Indifferent
	>= 6 years	Indifferent	Modified	Modified	Unmodified	Modified
Usage	< 20 hours/week	Indifferent	Modified	Indifferent	Indifferent	Modified
	>= 20 hours/week	Indifferent	Modified	Modified	Unmodified	Unmodified
Programming	None or Non Active	Indifferent	Modified	Modified	Unmodified	Modified
	Active Programming	Indifferent	Modified	Modified	Indifferent	Unmodified
Overall		Indifferent	Modified	Modified	Unmodified	Indifferent

Table 6-10 Code Preference based on Demographic Stratification

6.2 Survey Discussion

The indifference-based assessment of the 12 comparison questions indicated that in 4 comparison cases the participant exhibited indifference in their aesthetic preference, in 2 comparisons the participants preferred the unmodified code format and in 6 comparisons the participants preferred the modified code versions. The response, averaged over the 12 comparison questions, indicates an overall indifference to the aesthetic appearance of the code snippets. This overall indifference average is not of primary interest. The intent was not to develop a suite of reformatting techniques that would achieve the ultimate appearance. Of primary interest is the fact that some modifications did enhance the aesthetically appearances. These positive results imply that code can actually have its appearance aesthetically enhanced and opportunity for enhancement lies within the application of all three design principles. Listed below in Table 6-11 are the cases, along with the applied logic-based design principles, where the participant favoured the aesthetic appearance of the modified code.

Case	Principle - Modification	Snippet	Snippet Description
1	Balance - Indent	1	Minimal Indent
3	Balance - Indent	3	Moderate Indent
4	Balance - Margin	1	Minimal Indent
6	Balance - Margin	3	Moderate Indent
7	Rhythm - alignment	1	Minimal Indent
11	Unity - grouping	2	Deep Indent

Table 6-11 Cases where Modified Code Preferred

Balance is the distribution of optical weight within the frame (monitor or window). Increase in horizontal balance was investigated by increasing the indent depth (indent cases) and through the addition of a left margin (margin cases). The overall average response for the balanced-base modifications indicated an aesthetic preference for the modified code formats. This result, on its own, validates the main survey hypotheses of demonstrating that computer programming code, enhanced through the application of logic-based format modifications, could be more aesthetically pleasing than code formatted using standard GNU programming guidelines. The survey results also showed a more positive response for the graduated-indent scheme over the constant indent steps scheme.

The modifications associated with this principle of rhythm were referenced as alignment cases. They entailed manipulation of the code, where valid, to obtain an orderly repetition of elements; lines, shapes, tones, and textures. Case 7 indicated a preference for the modified code format. However, in the other two alignment cases (case

8 and case 9) participants preferred the appearance of the unmodified code format with the overall response indicating a preference for the appearance of the unmodified code. As one of the three rhythm-cases did show positive results, some potential merit may exist for the application of this design principle. Further study is warranted. The actual physical modifications were complicated and time consuming. Perhaps with a simpler subset of the manipulations, the overall response might be different.

The design principle of unity refers to the extent to which a group of visual elements are perceived as part of one single piece. Aesthetic appeal through the application of the design principle of unity was implemented by the grouping of similar constructs and by making the code appear more block-like. All global declarations were grouped at the top of each snippet of code. Flow control constructs were made more compact by moving the initializing brace to the same line as the key word. Long lines were split to create horizontally compact blocks of code.

Results for unity case 11 indicated a preference for the modified code format. In the other two unity-cases (Cases 10 and 12), participants showed indifference. The average response for the three cases response was indifference. Case 11 (preference for modified code) presented the most opportunity for modifications; many lines to be split and many braces to be moved to create more compact blocks. Cases 10 and 12 (indifferent to code options) provided little modification opportunity with the modified and unmodified versions looking fairly similar. These similar looking cases (snippets) were not, in retrospect, sufficiently good test material to evaluate the potential of applying the design principle of unity. Although not explicitly demonstrated, I feel that the design principle of unity did show potential for achieving enhanced aesthetic appeal.

The intent of this survey was to evaluate the responses assuming only the existence of one dependent variable (measured variable), aesthetic appeal. To verify this assumption demographic data was collected. As the demographic analysis was of a secondary importance, care and attention was not paid to adequately representing each of the strata. Under-filled strata were merged resulting in two strata per demographic factor. The demographic factors analysed were age, the number of years participants had been using a computer (history), the number of hours per week a participant used a computer (usage), and the exposure to programming (programming exposure).

The demographic analysis did demonstrate that the assumption of only one dependent variable, aesthetic appeal, was an oversimplification. Each of the demographic factors generated different perceived aesthetic appeal for their respective strata.

Sample participants over the age of 30 preferred the aesthetic appearance of the modified code format versions for all four modification techniques. On the other hand, the under 30 group showed a preference for the modified format only for the indent case. This younger group were neutral for the margin and grouping cases. They showed a strong preference for the unmodified format for the alignment case. The object of the demographic analysis was to ascertain if age is or is not a dependent variable. Understanding the cause of these differences in aesthetic preference is a subject for future study. The cause of the age based difference could be conjectured to be based on where participants primarily obtain information. The older group likely has spent a longer portion of their lives obtaining information from printed material. With printed material, typographical techniques have always been employed. Younger people probably turn to the computer as a data source. In this medium, aesthetic design principles are only now starting to be incorporated.

Similarly, the demographic factor of history of computer usage (history) displayed a difference in aesthetic appeal between the two strata. Both strata preferred the aesthetic appearance for the modified format in the indent cases with the participants with a shorter history demonstrating a more positive response. Both strata showed a similar response on the margin case but opposite responses on the alignment and grouping cases. What drives the fluctuation in the specific response is a subject for future study.

The aim of this thesis' research endeavour was to determine if these demographic factors actually had an influence on perceived aesthetic appeal. Indeed they do and will need to be more accounted for in any future studies. In particular the influence of the demographic factor of "exposure to programming" should be studied. Any potential benefit from creating a more aesthetic pleasing environment would stem from enhancing the performance of programmers. For this demographic factor, the evaluated strata represented active programmers and participants with no exposure of limited past exposure. The survey results for this demographic factor are shown below in Table 6-12.

Programming Strata	Normalized Response 12 Comparisons	Normalized Response for Modification Technique			
		Indent	Margin	Align	Grouping
None or Non Active	2.06	2.16	2.11	1.89	2.08
Active Programming	2.00	2.20	2.07	1.93	1.80
Overall	2.06	2.17	2.11	1.89	2.05

Programming Strata	Normalized Response 12 Comparisons	Interpretation per Modification Technique			
		Indent	Margin	Align	Grouping
None or Non Active	Indifferent	Modified	Modified	Unmodified	Modified
Active Programming	Indifferent	Modified	Modified	Indifferent	Unmodified
Overall	Indifferent	Modified	Modified	Unmodified	Indifferent

Table 6-12 Analyses for Demographic of Computer Exposure

The active programming group preferred the appearance of the indent cases and the margin cases. The indifference response for the alignment case indicated some potential may exist but requires more investigation. This active programming stratum did not like the "grouping" modification techniques. For the alignment (rhythm design principle) and grouping (unity design principle) the actual selection of applied reformatting techniques was fairly arbitrary and subject to my personal preferences. With an alternative set of modifications, the results might be different.

6.3 Conclusion

The main hypothesis of this survey was to address the question of whether or not computer programming code could be reformatted in order to increase its aesthetic appeal. The independent variable was the modification techniques applied to the code and the dependent variable was the perceived aesthetic preference of the survey participants. Four minor demographic based hypotheses were also investigated. The demographic hypotheses investigated the effect of age, the number of years participants had been using a computer (history), the number of hours per week a participant used a computer (usage), and the exposure to programming (programming exposure). Four snippets of code were modified using the design principles of balance (indent and margin), rhythm and unity. The responses of the survey participant were analysed using an 'indifference analysis'. The conclusions reached were:

1. This survey showed evidence that through the application of the design principle of balance, the appearance of computer code could be aesthetically enhanced.

   In six cases of the eight balance cases (margin and indent), participants found the modified code format more aesthetically pleasing. In the other two balance cases, they were indifferent with preference. The balance design principle was applied by increasing the level of indentation and adding a left margin.

2. The results associated with the rhythm design principle indicated that the participant directionally preferred the aesthetic appearance of the unmodified code. This design principle, as applied in the survey, did not effect an enhancement to the aesthetic appeal of the selected code snippets.

   Code modifications associated with this principle were complicated and time consuming. With a different set of modifications, a different response may have been obtained. In retrospect, my selection of modifications may not have been adequate.

3. The results associated with the unity design principle indicated that the participants were indifferent overall, however, participants directionally preferred the aesthetic appearance of the modified code for the key snippet.

   Only one code snippet provided adequate scope to apply the unity principle with the other two being inappropriate. The inappropriate snippets generated an indifference response. Further investigation with other code snippets is required.

4. The assumption of the existence of only one dependent variable, aesthetic appeal, was an oversimplification. The demographic factors of age, the number of years participants had been using a computer, the number of hours per week a participant used a computer, and the exposure to programming should be considered dependent variables and need to be factored into any future work.

   The demographic factors of age and history generated different overall strata responses. Both strata for the factors of usage and programming presented indifference responses.

5. This survey has provided positive incentive to justify further study into the approach of using Gestalt based principles as a means of enhancing the aesthetic appeal of computer code.

   Computer code had its aesthetic appeal enhanced through the application of at least the balance design principle and potentially through the unity principle. Researchers like Ngo and Byrne [47, 48] , Ngo et al. [49, 76], and Tullis [6, 7, 50, 51, 78] developed and validated mathematical models to quantify the aesthetic appeal of web-pages and alphanumeric displays. Adaptations of these models could be useful to quantify the aesthetic appeal of reformatted code prior to attempting to verify these results.

6. Can computer code be aesthetically enhanced? This survey provided directionally evidence that the appearance of computer code can be enhanced. It also demonstrated the potential of applying a logic-based approach, Gestalt design principles, as an approach to achieving these enhancements.

6.4 Chapter Summary

This chapter provided the analyses, discussion and conclusions associated with the user preference survey. Forty individuals participated in the user study. Because the true distribution for the population was not known and because the mean ± the standard deviation nearly matched the range of the employed scale, standard statistical analyses could not provide insight. To address this statistical analyses dilemma, the question responses were evaluated through a technique I dubbed 'indifference analyses. A normalized response that fell within ± two standard deviations of the indifference point (random response to the question) was deemed to indicate no preference. Outside this range the response indicated a preference, the preference being modified or unmodified depending of relative positioning. Table 2 and Table 3 summarise this analysis for the 13 comparisons. In six comparison cases, the participants directionally preferred the modified versions of the code snippets, in three cases they preferred the unmodified versions, and in four cases they showed indifference. The modified code choices were associated with the indent and margin modifications; the design principle of balance.

This survey provided evidence that the appearance of code could be enhanced through the application of the design principle of balance and potentially by unity. Greater horizontal balance was achieved through increasing the indent depth and through including a left margin. In some of the individual cases associated with these later design principles, positive responses were obtained. These positive responses provide enough optimism to warrant further investigation.

The analyses of the gathered demographic data indicate that these demographic factors influence the perceived aesthetic appeal of code. The demographic factors investigated were age, how long a participant used a computer, the number of hours of use per week, and exposure to programming. Even though each of these factors demonstrated different responses between their strata, the demographic analysis only provided directional insights. The selection of participant was on a first come basis with no attempt to ensure the strata for each of the demographic factors was adequately represented. The survey did however establish that these factors are significant and need to be addressed more thoroughly in any future work.

The main conclusion of the survey was that there is evidence that appearance of computer code can be enhanced. This enhancement was achieved through the application design principle of balance.

CHAPTER SEVEN: THESIS SUMMARY AND CONTIBUTIONS

Can programming code be presented in a more aesthetically pleasing fashion? Is there any benefit to reformatting computer code? To address these questions, I developed a basic understanding of the meaning of "aesthetics" through an extensive investigation into related work covering a number of disciplines (philosophy, typography, psychology, graphic design, and marketing), each with their own vocabulary. To distil this information and relate it to computer code was a nontrivial exercise.

Chapter 2 researched the areas of aesthetics and aesthetic affect, providing the necessary foundation to investigate the feasibility of effecting aesthetic modifications to computer code. The historic and current practice of addressing aesthetic affect (mood) within the field of computer science (hardware design, algorithm design, and web-page design) was examined in Chapter 3. Chapter 4 looked at the work done by web-page designers and alpha-numeric display researchers to develop aesthetically pleasing appearances; as well, it looked at design practices within the field of typography. The overlap in design philosophies of these two separate areas provided the rationale for the selection of the logic-based code reformatting methodology: the application of the Gestalt principles of balance, rhythm and unity. The thesis research question was addressed through the development, implementation, and analysis of a user-preference survey. Chapter 5 outlines the survey design and implementation. It also addresses the selection of the code snippets and the application of logic-based modification techniques. In Chapter 6 the survey data was analysed, discussed, and conclusions were drawn.

The main contribution of this thesis was to provide evidence that appearance of computer code can be aesthetically enhanced. This evidence was demonstrated through the application design principle of balance. Survey participants preferred the aesthetic appearance of that code snippets modified by increasing the depth of indentation and by including a left margin.

A second thesis contribution was the development of a code modification scheme based on the Gestalt design principles. This approach was based on the work of web-site and alphanumeric design researchers. Only the design principle of balance provided definitive results, but the potential of applying other rules was established.

This thesis also established that demographic factors such as age, how long a participant used a computer, the number of hours of use per week, and exposure to programming need to be incorporated into any future studies.

To advance this line of study, further work on methods to establish and enhance aesthetic appeal needs to be undertaken. The results of this thesis, that aesthetic appeal could be advanced through the application of the design principle of balance, needs to be confirmed. Further work with regard to utilizing the design principles of rhythm and unity needs to be investigated. Work to investigate the aesthetic enhancement potential of the other overlap design rules; homogeneity / symmetry and proportion, should be considered.

In addition, other logic-based systematic approaches need to be identified and tested. In this work only the 'C' programming language was investigated. Enhancing the appearance of other programming language could be explored.

If further work does verify that computer code's appearance can be aesthetically enhanced, then potential positive benefits could be analysed. Work could be undertaken to demonstrate that the programmer's frame of mind has been altered; a change in aesthetic affect. The benefits of this altered mood could be quantified. Some of these benefits would be higher levels of focus, faster understanding or comprehension, fewer mistake, and faster identification of mistakes. These benefits would lead directly to an enhancement of the programmers' performance.

Enhanced programmer performance could result in an overall reduction in the cost of programming and program maintenance. Programs would also reach the markets at a faster rate. These programming cost savings would translate into savings for the target clientele, the end users. This thesis provided the initial step to the realization of these economic benefits.

REFERENCES

1. Kernighan, B.W. and Plauger, P.J., *The Elements of Programming Style*. 1974: McGraw-Hill Book Company.

2. Kurosu, M. and Kashimura, K. *Apparent Usability vs. Inherent Usability*. in *CHI '95 Conference Companion*. 1995.

3. Tractinsky, N. *Aesthetics and Apparent Usability: Empirically Assessing Cultural and Methodological Issues*. in *CHI '97*. 1997. Atlanta GA.

4. Toh, S.C., *Cognitive and Motivational Effects of Two Multimedia Simulation presentations Modes on Science Learning*, University of Science Malaysia, Doctoral dissertation, 1998.

5. Aspillage, M., *Screen Design: A location of Information and Its Effects on Learning*. Journal of Computer-Based Instruction, 1991. 18(3): p. 89-92.

6. Tullis, T.S., *An Evaluation of Alphanumeric, Graphic, and Color Information Displays*. Human Factors, 1981. 23: p. 541-550.

7. Tullis, T.S., *Predicting the Usability of Alphanumeric Displays*, Rice University, Doctoral dissertation, 1984.

8. Keister, R.W. and Gallaway, G.R. *Making Software User Friendly: An Assessment of Data Entry Performance*. in *Human Factors Society 27th Annual Meeting Proceedings*. 1983. Santa Monica, CA.

9. Linnenbrink, E. and Pintrich, P., *Role of Affect in Cognitive Processing in Academic Contexts*, in *Motivation, Emotion and Cognition: Integrative Perspectives on Intellectual Functioning and Development*, D. Yun Dai, R.J. Sternberg, and R.J. Erlbaum, Editors. 2004, Lawrence Erlbaum Associates. p. 57-88.

10. Oman, P.W. and Cook, C.R., *A Taxonomy for Programming Style*, in *Proceedings of the 1990 ACM annual conference on Cooperation*. 1990, ACM Press: Washington, DC. p. 244-250.

11. Soanes, C. and Hawker, S., eds. *Compact Oxford English Dictionary of Current English*. Third ed. 2005, Oxford University Press.

12. *Aesthetics*. Microsoft Encarta Online Encyclopedia [cited 2005 February]; Available from: http://uk.encarta.msn.com/encyclopedia_761576304/Aesthetics.html.

13. *Aesthetics*. The Internet Encyclopedia of Philosophy [Web-site: University of Western Australia] [cited 2005 March]; Available from: http://www.iep.utm.edu/a/aestheti.htm.

14. Carroll, N., *Philosophy of Art: A Contemporary Introduction*. Routledge Contemporary Introductions to Philosophy, ed. Routledge. 1999, London.

15. Fisher, J. *Course: Philosophy 3700 - Aesthetic Theory - Fall 2004*. [Web Site: University Colorado] [cited 2005 February]; Available from: http://spot.colorado.edu/~jafisher/3700f04/.

16. Rowlands, J. and Landauer, J. *Importance of Philosophy*. [Online Book] [cited 2005 February]; Available from: http://www.importanceofphilosophy.com.

17. Hartmann, J. and Sutcliffe, A. *Aesthetic Judgment of Interactive Systems*. in *CHI2005 Workshop: Understanding and Designing for Aesthetic Experience*. 2006. Edinburgh.

18. Lavie, T. and Tractinsky, N., *Assessing dimensions of perceived visual aesthetics of web sites*. International Journal of Human-Computer Studies, 2004. 60(3): p. 269-298.

19. Norman, D.A., *Emotional Design: Why we love (or Hate) Everyday Things*. 2004, New York: Basic Books.

20. Norman, D.A., *The Design of Everyday Things*. 1990, New York: Doubleday.

21. Li, N. and Zhang, P. *A Research Agenda toward Assessing Perceived Affective Quality of IT*. in *Proceedings of the Eleventh Americas Conference on Information Systems*. 2005. Omaha NE.

22. Zhang, P. and Li, N., *The Importance of Affective Quality*. Communications of the ACM, 2005. 48(9): p. 105-108.

23. Lindgaard, G., *Aesthetics, visual appeal, usability, and user satisfaction: What do the user's eyes tell the user's brain?* in *Hot Topics 6(5)*. 2006, HCI News and Ideas from the Human Oriented Technology Lab at Carleton University: Ottawa.

24. Pham, M.T., Cohen, M.T., Pracejus, J.W., and Hughes, G.D., *Affect monitoring and the primacy of feelings in judgment*. Journal of Consumer Research, 2001. 28((2001)): p. 167-188.

25. Duckworth, K., Bargh, J.A., Garcia, M., and Chaiken, S., *The automatic evaluation of novel stimuli*. Psychological Science, 2002. 13(6): p. 513-519.

26. Chang, D., Dooley, L., and Tuovinen, J., E., *Gestalt theory in visual screen design: a new look at an old subject*, in *Proceedings of the Seventh world*

conference on computers in education conference on Computers in education: Australian topics - Volume 8. 2002, Australian Computer Society, Inc.: Copenhagen, Denmark.

27. Liu, Y., *Engineering aesthetics and aesthetic ergonomics: Theoretical foundations and a dual-process research methodology*. Ergonomics, 2003. 46(13/14): p. 1273-1292.

28. Postrel, V., *The Substance of Style*. 2002: Harper Collins.

29. Tractinsky, N., *Does Aesthetics Matter in Human-Computer Interaction?* in *Mensch Computer: Humans & Computer 2005 Art and Science-trespass beyond the border of the interactive KIND*. 2005: Johannes Kepler University of Linz.

30. Hoffmann, R. and Krauss, K. *A Critical Evaluation of Literature on Visual Aesthetics for the Web*. in *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. 2004. Stellenbosch, Western Cape, South Africa: ACM International Conference Proceeding Series.

31. Infoplease.com. *Computer Usage in the U.S.* [cited 2006 November]; Available from: http://www.infoplease.com/ipa/A0921872.html.

32. Tractinsky, N., *Toward the Study of Aesthetics in Information Technology*, in *Proceedings of the 25th Annual International Conference on Information Systems (ICIS)*. 2004: Washington, DC. p. 771-780.

33. Budnick, P. *Commentary: What is Ergonomics Really About?* Ergonweb: News and Information May 30, 2001 [cited 2007 November]; Available from: http://www.ergoweb.com/news/detail.cfm?id=345.

34. Liu, Y., *The aesthetic and the ethic dimensions of human factors and design*. Ergonomics, 2003. 46(13/14): p. 1293-1305.

35. Myers, B.A., *A Brief History of Human-Computer Interaction Technology*. ACM Interactions, 1998. 5(2): p. 44-45.

36. Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong, and Verplank. *Human Computer Interaction*. ACM SIGCHI Curricula for Human-Computer Interaction 2004 [cited 2006 November]; Available from: http://sigchi/cdg/cdg2.html.

37. Burmester, M., Platz, A., Rudolph, U., and Wild, B., *Aesthetic design - just an add on?* in *Proceedings of HCI International (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Ergonomics*

and User Interfaces-Volume I - Volume I. 1999, Lawrence Erlbaum Associates, Inc.

38. De Angeli, A., Sutcliffe, A., and Hartmann, J. *Interaction, usability and aesthetics: what influences users' preferences?* in *Proceedings of the 6th ACM Conference on Designing interactive Systems*. 2006. University Park PA: ACM Press.

39. Graves Petersen, M., Iversen, O.S., Krogh, P.G., and Ludvigsen, M., *Aesthetic interaction: a pragmatist's aesthetics of interactive systems*, in *Proceedings of the 2004 conference on Designing interactive systems: processes, practices, methods, and techniques*. 2004, ACM Press: Cambridge MA.

40. Tractinsky, N., Katz, A.S., and Ikar, D., *What is beautiful is usable.* Interacting with computers, 2000. 13(2): p. 127-146.

41. Krauss, K., *Visual aesthetics and its effect on communication intent: a theoretical. study and website evaluation*, in *Proceedings of the Southern African Computer Lecturers Association (SACLA)*. 2004. p. 451-469.

42. Nakarada-Kordic, I. and Lobb, B., *Effect of perceived attractiveness of web interface design on visual search of web sites*, in *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*. 2005, ACM Press: Auckland, New Zealand.

43. Jordan, P.W., *Human factors for pleasure in product use.* Applied Ergonomics, 1998. 29(1): p. 25-33.

44. Light, A., *Report: Aesthetic Approaches to HCI*, in *Usability News*. 2005.

45. De Angeli, A., Lynch, P., and Johnson, G.I., *Pleasure versus efficiency in user interfaces: Towards an involvement framework*, in *Pleasure with products: Beyond usability*, W.S. Green and P.W. Jordan, Editors. 2002, Taylor & Frances. p. 97-111.

46. Hassenzahl, M., Platz, A., Burmester, M., and Lehner, K., *Hedonic and ergonomic quality aspects determine a software's appeal*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2000, ACM Press: The Hague, Netherlands.

47. Ngo, D.C.L. and Byrne, J.G. *Aesthetic Measures for Screen Design.* in *Proceedings of the Australasian Conference on Computer Human Interaction.* 1998: IEEE Computer Society.

48. Ngo, D.C.L. and Byrne, J.G., *Another look at a Model for Evaluating Interface Aesthetics*. International Journal of Applied Mathematics and Computer Science, 2001. 11(2): p. 515-535.

49. Ngo, D.C.L., Teo, L.S., and Byrne, J.G., *Modeling interface aesthetics*. Information Sciences: an International Journal, 2003. 152(1): p. 25-46.

50. Tullis, T.S. *A Computer-Based Tool for Evaluating Alphanumeric Displays*. in *Proceedings of INTERACT'84 Conference on Human-Computer Interaction*. 1984. London: North-Holland.

51. Tullis, T.S., *Optimizing the usability of computer-generated displays*, in *Proceedings of the Second Conference of the British Computer Society, human computer interaction specialist group on People and computers: designing for usability*. 1986, Cambridge University Press: York, United Kingdom.

52. Knuth, D.E., *Computer Programming as an Art*. Communications of the ACM, 1974. 17(12): p. 667-673.

53. Hoare, C.A.R., Hayes, I.J., Jifeng, H., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M., and Sufrin, B.A., *Laws of programming*. Communications of the ACM, 1987. 30(8): p. 672-686.

54. Strunk, W. and White, E.B., *Elements of Style*. 1959, New York: MacMillan.

55. Weissman, L., *Psychological complexity of computer programs: an experimental methodology*. ACM SIGPLAN Notices, 1974. 9(6): p. 25-36.

56. Schach, S.R., *Classical and Object-Oriented Software Engineering with UML and Java*. 1999, The McGraw-Hill Companies, Inc. p. 9-11.

57. Dijkstra, E.W., *A Case Against the GoTo Statement*. Communications of the ACM, 1968. 11(3): p. 147-148.

58. Burley, C. *General Programming Rules*. [Web-site: a collaboration of the center for the public domain and University of Northern Carolina-ch] [cited 2005 April]; Available from: http://www.ibiblio.org/pub/languages/fortran/ch1-3.html.

59. Parks, D. *Programming Style Guidelines*. [Web-site: Appalachian State University] [cited 2005 April]; Available from: http://www.cs.appstate.edu/u/cs/dap/style.html.

60. Kernighan, B.W. and Pike, R., *The Practice of Programming*. 1999: Addison-Wesley.

61. Kernighan, B.W. and Plauger, P.J., *The Elements of Programming Style*. 2nd ed. 1978: McGraw-Hill Book Company.

62. Kreitzberg, C.B. and Shneiderman, B., *The Elements of FORTRAN Style: Techniques for Effective Programming*. 1972: Harcourt Brace College Publishers.

63. Burley, C., Vajhoej, A., Page, C., and Plotkin, K. *Program Layout - The Art of Making Programs Readable*. [Web-site: a collaboration of the center for the public domain and University of Northern Carolina-ch] [cited 2005 April]; Available from: http://www.ibiblio.org/pub/languages/fortran/ch1-6.html.

64. Clifton, M.H., *A technique for making structured programs more readable*. SIGPLAN Not., 1978. 13(4): p. 58-63.

65. Horton, I., *Beginning C*. 2nd ed. Wrox Beginning Series. 1997: Wrox Press Ltd.

66. Shneiderman, B. and McKay, D., *Experimental Investigations of computer program debugging and modifications*, in *6th International Congress of the International Ergonomics Association*. 1976.

67. Leinbaugh, D.W., *Indenting for the compiler*. ACM SIGPLAN Notices, 1980. 15(5): p. 41-48.

68. Miara, R.J., Musselman, J.A., Navarro, J.A., and Shneiderman, B., *Program indentation and comprehensibility*. Communications of the ACM, 1983. 26(11): p. 861-867.

69. Krall, A. and Harris, W., *An investigation of programming style on readability / understanding of a simple COBOL program; The effect of indentation and vertical spacing*. 1980, University of Maryland.

70. Love, T., *An experimental investigation of the effect of program structure on program understanding*, in *Proceedings of an ACM conference on Language design for reliable software*. 1977: Raleigh, North Carolina.

71. Oman, P.W. and Cook, C.R., *Typographic style is more than cosmetic*. Communications of the ACM, 1990. 33(5): p. 506 - 520.

72. Baecker, R., *Enhancing program readability and comprehensibility with tools for program visualization*, in *Proceedings of the 10th international conference on Software engineering*. 1988, IEEE Computer Society Press: Singapore. p. 356 - 366.

73. Baecker, R.M. and Marcus, A., *Human Factors and Typography for More Readable Programs*. 1990: Addison Wesley Publishing Company.

74. Leavens, G.T., *Prettyprinting Styles for Various Languages*. ACM SIGPLAN Notices, 1984. 19(2).

75. Ruckert, M., *Conservative pretty printing.* ACM SIGPLAN Notices, 1997. 32(7): p. 39-44.

76. Ngo, D.C.L., Teo, L.S., and Byrne, J.G., *A Mathematical Theory of Interface Aesthetics.* Visual Mathematics, 2000. 2(4).

77. Reilly, S.S. and Roach, J.W., *Improved Visual Design for Graphic Displays.* IEEE Computer Graphics and Applications, 1984. 4(2): p. 42-51.

78. Tullis, T.S., *The formatting of alphanumeric displays: A review and analysis.* Human Factors, 1983. 25: p. 657-682.

79. Wertheimer, M. *Gestalt Theory.* [Address before the Kant Society, Berlin 1924, from The Gestalt Archive, maintained by the Society for Gestalt Theory and its Application] [cited 2006 April]; Available from: http://www.gestalttheory.net/archive/wert1.html.

80. Parizotto-Ribeiro, R. and Hammond, N. *What is Aesthetics anyway? investigating the use of the design principles.* in *Aesthetic Approach to Human-Computer Interaction, NordCHI 2004 Workshop.*

81. Parizotto-Ribeiro, R. and Hammond, N. *Does Aesthetics Affect the Users' Perceptions of VLEs?* in *The 12th International Conference on Artificial Intelligence in Education.* 2005. Amsterdam.

82. Turnbull, A.T. and Baird, R.N., *The Graphics of Communication.* Fourth ed. 1980: Holt, Rinehart and Winston.

83. Burt, C., *A Psychological Study of Typography.* 1959: Cambridge University Press.

84. Collier, D., *Collier's Rules for Desktop Design and Typography.* 1991: Addison Wesley.

85. Ozubko, C. *Graphic Design Awareness in a Computing Environment.* in *Putting it all together - SIGUCCS XIII.* 1985: ACM.

86. Galizt, W., *Handbook of Screen Format Design.* 1981: QED Publishing Group.

87. Galizt, W., *Essential guide to user interface design: An Introduction to GUI Principles and Techniques.* 1996: Wiley Computer Publishing.

88. Galizt, W. and DiMatteo, A., *EIS forms and screens design manuals.* INA Technical Reports. 1974: INA Corporation.

89. Berleant, D., *On Site: does typography affect proposal assessment?* Communications of the ACM, 2000. 43(8): p. 24-25.

90. Harrington, S.J., Naveda, J.F., and Jones, R.P., *Aesthetic measures for automated document layout*, in *ACM Symposium on Document Engineering*. 2004, ACM Press.

91. *SimpleScalar LLC*. 2006 [cited; Available from: http://www.simplescalar.com/.

92. Baecker, R.M. and Marcus, A., *Visualizing C: The Graphic Design and Electronic Publishing of C Programs and Their Documentation*. 1989: Addison Wesley.

93. Love, T., *Relating Individual Differences in Computer Programming Performance to Human Information Processing Abilities*, University of Washington, Doctoral dissertation, 1977.

94. *Research Randomizer*. [cited 2006 May]; Available from: http://www.randomizer.org/.

95. *Central Limits Theorem*. Statistical Engineering Website: statistical definitions [cited 2007 June]; Available from: http://www.statisticalengineering.com/central_limit_theorem.htm.

APPENDIX B: CODE SNIPPETS

Figures	Modification	Snippet
B-1	Base 1	Snippet 1
B-2	Base 2	Snippet 2
B-3	Base 3	Snippet 3
B-4	Base 4	Snippet 4
B-5	Indent 1	Snippet 1
B-6	Indent 2	Snippet 2
B-7	Indent 3	Snippet 3
B-8	Indent 4	Snippet 4
B-9	Margin 1	Snippet 1
B-10	Margin 2	Snippet 2
B-11	Margin 3	Snippet 3
B-12	Align 1	Snippet 1
B-13	Align 2	Snippet 2
B-14	Align 3	Snippet 3
B-15	Group 1	Snippet 1
B-16	Group 2	Snippet 2
B-17	Group 3	Snippet 3

Table B-1 List of Code Snippet Figures in Appendix B

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_tnext_PC;
    md_addr_tpred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[2])
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

Figure B-1 Base 1 - Snippet 1

```
static void ruu_writeback(void)
{
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event()))
    {
       rs->completed = TRUE;
       if (rs->recover_inst)
       {
          ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, rs->recover_inst ? PEV_MPDETECT : 0);
       }
       for (i=0; i<MAX_ODEPS; i++)
       {
          if ( rs->onames[i] != NA )
          {
             if ( rs->spec_mode )
             {
                link = spec_create_vector[rs->onames[i]];
                if (link.rs && (link.rs == rs && link.odep_num == i))
                {
                   spec_create_vector[rs->onames[i]] = CVLINK_NULL;
                   spec_create_vector_rt[rs->onames[i]] = sim_cycle;
                   spec_create_vector_xt[rs->onames[i]] = cycle;
                }
             }
             else
             {
                link = create_vector[ rs->onames[i]];
                if (link.rs && (link.rs == rs && link.odep_num == i) )
                {
                   create_vector[ rs->onames[i]]=CVLINK_NULL;
                   create_vector_rt[ rs->onames[i]]=sim_cycle;
                   create_vector_xt[rs->onames[i]] = cycle;
                }
             }
             rs->odep_list[i]= NULL;
          }
       }
    }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
static md_addr_t fetch_pred_PC;
static struct fetch_rec *fetch_data;
static int fetch_num;
static int fetch_tail, fetch_head;
```

Figure B-2 Base 2 - Snippet 2

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev)
    {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_init(void)
{
    sim_num_refs = 0;
    regs_init(®s);
    mem = mem_create("mem");
    mem_init(mem);
}

static char * simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t *val);
static char * simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p,int nbytes);
static char * simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
    ld_load_prog(fname, argc, argv, envp, ®s, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Figure B-3 Base 3 - Snippet 3

```
int i, j, index, n_std_unknowns;
md_addr_t std_unknowns[MAX_STD_UNKNOWNS];
for (i=0, index=LSQ_head, n_std_unknowns=0;i < LSQ_num; i++, index=(index + 1) % LSQ_size)
{
    if ( (MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE))
    {
        if (!STORE_ADDR_READY(&LSQ[index]))
        {
            break;
        }
        else if (!OPERANDS_READY(&LSQ[index]))
        {
            if (n_std_unknowns == MAX_STD_UNKNOWNS)
                fatal("STD unknown array overflow, increase MAX_STD_UNKNOWNS");
            std_unknowns[n_std_unknowns++] = LSQ[index].addr;
        }
        else
        {
            for (j=0; j < n_std_unknowns; j++)
            {
                if (std_unknowns[j] == LSQ[index].addr)std_unknowns[j] = 0;
                if (rs->in_LSQ && ((MD_OP_FLAGS(rs->op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE)))
                {
                    rs->issued = TRUE;
                    rs->completed = TRUE;
                    if (rs->onames[0] || rs->onames[1])
                    {
                        panic("mis-predicted store");
                        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, 0);
                        n_issued++;
                    }
                }
            }
        }
    }
    if ( ((MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_LOAD)) == (F_MEM|F_LOAD))&& !LSQ[index].queued
        && !LSQ[index].issued && !LSQ[index].completed && OPERANDS_READY(&LSQ[index]))
    {
        for (j=0; j
        {
            if (std_unknowns[j] == LSQ[index].addr) break;
        }
        if (j == n_std_unknowns)
        {
            readyq_enqueue(&LSQ[index]);
        }
    }
}
```

Figure B-4 Base 4 - Snippet 4

```
void sim_uninit(void)
{
        if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
        md_inst_t IR;
        enum md_opcode op;
        md_addr_t PC;
        md_addr_tnext_PC;
        md_addr_tpred_PC;
        int ea_comp;
        int in_LSQ;
        int recover_inst;
        int stack_recover_idx;
        struct bpred_update_t dir_update;
        int spec_mode;
        md_addr_t addr;
        INST_TAG_TYPE tag;
        INST_SEQ_TYPE seq;
        unsigned int ptrace_seq;
        int slip;
        int queued;
        int issued;
        int completed;
        int onames[MAX_ODEPS];
        struct RS_link *odep_list[MAX_ODEPS];
        int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[2])
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
        RUU = calloc(RUU_size, sizeof(struct RUU_station));
        if (!RUU) fatal("out of virtual memory");
        RUU_num = 0;
        RUU_head = 0;
        RUU_tail = 0;
        RUU_count = 0;
        RUU_fcount = 0;
}
```

Figure B-5   Indent 1 - Snippet 1

```
static void ruu_writeback(void)
{
        int i;
        struct RUU_station *rs;
        while ((rs = eventq_next_event()))
        {
            rs->completed = TRUE;
            if (rs->recover_inst)
            {
                ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
                    rs->recover_inst ? PEV_MPDETECT : 0);
            }
            for (i=0; i<MAX_ODEPS; i++)

                if ( rs->onames[i] != NA )
                {
                    if ( rs->spec_mode )
                    {
                        link = spec_create_vector[rs->onames[i]];
                        if (link.rs && (link.rs == rs && link.odep_num == i)
                        {
                            spec_create_vector[rs->onames[i]] = CVLINK_NULL;
                            spec_create_vector_rt[rs->onames[i]] = sim_cycle;
                            spec_create_vector_xt[rs->onames[i]] = cycle;
                        }
                    }
                    else
                    {
                        link = create_vector[ rs->onames[i]];
                        if (link.rs && (link.rs == rs && link.odep_num == i) )
                        {
                            create_vector[ rs->onames[i]]=CVLINK_NULL;
                            create_vector_rt[rs->onames[i]] = sim_cycle;
                            create_vector_xt[ rs->onames[i]]= cycle;
                        }
                    }
                    rs->odep_list[i]= NULL;
                }
            }
        }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
        struct spec_mem_ent *next;
        md_addr_t addr;
        unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
static md_addr_t fetch_pred_PC;
static struct fetch_rec *fetch_data;
static int fetch_num;
static int fetch_tail, fetch_head;
```

Figure B-6 Indent 2 - Snippet 2

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
        struct RS_link *prev;
        struct RS_link *ev;
        tick_t when = 0;
        ev = event_queue;
        while(ev)
        {
                if(ev->rs == rs) return ev->x.when;
                ev = ev->next;
        }
        return when;
}

void sim_init(void)
{
        sim_num_refs = 0;
        regs_init(®s);
        mem = mem_create("mem");
        mem_init(mem);
}

static char * simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t *val);
static char * simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p,int nbytes);
static char * simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
        ld_load_prog(fname, argc, argv, envp, ®s, mem, TRUE);
        if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
        else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
        fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
        rslink_init(MAX_RS_LINKS);
        tracer_init();
        fetch_init();
        cv_init();
        eventq_init();
        readyq_init();
        ruu_init();
        lsq_init();
        dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Figure B-7 Indent 3 - Snippet 3

```
int i, j, index, n_std_unknowns;
md_addr_t std_unknowns[MAX_STD_UNKNOWNS];
for (i=0, index=LSQ_head, n_std_unknowns=0;i < LSQ_num; i++, index=(index + 1) % LSQ_size)
{
        if ( (MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE))
        {
                if (!STORE_ADDR_READY(&LSQ[index]))
                {
                        break;
                }
                else if (!OPERANDS_READY(&LSQ[index]))
                {
                        if (n_std_unknowns == MAX_STD_UNKNOWNS)
                                fatal("STD unknown array overflow, increase MAX_STD_UNKNOWNS");
                        std_unknowns[n_std_unknowns++] = LSQ[index].addr;
                }
                else
                {
                        for (j=0; j < n_std_unknowns; j++)
                        {
                                if (std_unknowns[j] == LSQ[index].addr)std_unknowns[j] = 0;
                                if (rs->in_LSQ && ((MD_OP_FLAGS(rs->op) &
                                        (F_MEM|F_STORE)) == (F_MEM|F_STORE)))
                                {
                                        rs->issued = TRUE;
                                        rs->completed = TRUE;
                                        if (rs->onames[0] || rs->onames[1])
                                        {
                                                panic("mis-predicted store");
                                                ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, 0);
                                                n_issued++;
                                        }
                                }
                        }
                }
        }
        if ( ((MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_LOAD)) == (F_MEM|F_LOAD))&& !LSQ[index].queued
                && !LSQ[index].issued && !LSQ[index].completed && OPERANDS_READY(&LSQ[index]))
        {
                for (j=0; j
                {
                        if (std_unknowns[j] == LSQ[index].addr) break;
                }
                if (j == n_std_unknowns)
                {
                        readyq_enqueue(&LSQ[index]);
                }
        }
}
```

Figure B-8 Indent 4 - Snippet 4

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_tnext_PC;
    md_addr_tpred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS) (RS)->idep_ready[0]  \
    && (RS)->idep_ready[1] && (RS)->idep_ready[2])
#define OPERANDS_COMPT(RS) (RS)->idep_compt[0]  \
    && (RS)->idep_compt[1] && (RS)->idep_compt[2])
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

Figure B-9 Margin 1 - Snippet 1

```
static void ruu_writeback(void)
{
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event()))
    {
      rs->completed = TRUE;
      if (rs->recover_inst)
      {
         ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
           rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
         if ( rs->onames[i] != NA )
         {
           if ( rs->spec_mode )
           {
             link = spec_create_vector[rs->onames[i]];
             if (link.rs && (link.rs == rs && link.odep_num == i))
             {
               spec_create_vector[rs->onames[i]] = CVLINK_NULL;
               spec_create_vector_rt[rs->onames[i]] = sim_cycle;
               spec_create_vector_xt[rs->onames[i]] = cycle;
             }
           }
           else
           {
             link = create_vector[ rs->onames[i]];
             if (link.rs && (link.rs == rs && link.odep_num == i) )
             {
               create_vector[ rs->onames[i]]=CVLINK_NULL;
               create_vector_rt[ rs->onames[i]]=sim_cycle;
               create_vector_xt[rs->onames[i]] = cycle;
             }
           }
           rs->odep_list[i]= NULL;
         }
      }
    }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
static md_addr_t fetch_pred_PC;
static struct fetch_rec *fetch_data;
static int fetch_num;
static int fetch_tail, fetch_head;
```

Figure B-10 Margin 2 - Snippet 2

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev)
    {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_init(void)
{
    sim_num_refs = 0;
    regs_init(®s);
    mem = mem_create("mem");
    mem_init(mem);
}

static char * simoo_reg_obj(struct regs_t *regs, int is_write,
    enum md_reg_type rt, int reg, struct eval_value_t *val);
static char * simoo_mem_obj(struct mem_t *mem, int is_write,
    md_addr_t addr, char *p,int nbytes);
static char * simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs,
    struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
    ld_load_prog(fname, argc, argv, envp, ®s, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Figure B-11 Margin 3 - Snippet 3

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

#define MAX_IDEPS    3
#define MAX_ODEPS    2
#define OPERANDS_READY(RS)    \
        (RS)->idep_ready[0]    &&   \
        (RS)->idep_ready[1]    &&   \
        (RS)->idep_ready[2])
#define OPERANDS_COMPT(RS)    \
        (RS)->idep_compt[0]    &&   \
        (RS)->idep_compt[1]    &&   \
        (RS)->idep_compt[2])
typedef unsigned int   INST_TAG_TYPE;
typedef unsigned int   INST_SEQ_TYPE;
struct  RUU_station {
    int   slip;
    int   queued;
    int   issued;
    int   in_LSQ;
    int   ea_comp;
    int   completed;
    int   spec_mode;
    int   recover_inst;
    int   stack_recover_idx;
    int   onames[MAX_ODEPS];
    int   idep_ready[MAX_IDEPS];
    md_inst_t     IR;
    md_addr_t     PC;
    md_addr_t     addr;
    md_addr_t     next_PC;
    md_addr_t     pred_PC;
    unsigned int  ptrace_seq;
    struct RS_link  *odep_list[MAX_ODEPS];
    enum md_opcode        op;
    INST_TAG_TYPE        tag;
    INST_SEQ_TYPE       seq;
    struct bpred_update_t   dir_update;
};
static int      RUU_num;
static int      RUU_head;
static int      RUU_tail;
static struct   RUU_station *RUU;

static void ruu_init(void)
{
    RUU_tail      = 0;
    RUU_num       = 0;
    RUU_head      = 0;
    RUU_count     = 0;
    RUU_fcount    = 0;
    RUU   =   calloc(RUU_size,   sizeof(struct RUU_station));
    if (!RUU)    fatal("out of virtual memory");
}
```

Figure B-12   Align 1 - Snippet 1

```
static void ruu_writeback(void)
  {
  int  i;
  struct RUU_station  *rs;
  while ((rs  =  eventq_next_event()))
  {
     rs->completed  =  TRUE;
     if (rs->recover_inst)
     {
        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
                 rs->recover_inst ? PEV_MPDETECT : 0);
     }
     for (i=0; i<MAX_ODEPS; i++)
     {
        if ( rs->onames[i] != NA )
        {
           if ( rs->spec_mode )
           {
              link = spec_create_vector[rs->onames[i]];
              if (link.rs && (link.rs == rs && link.odep_num == i))
              {
                 spec_create_vector_xt[rs->onames[i]] = cycle;
                 spec_create_vector_rt[rs->onames[i]]  = sim_cycle;
                 spec_create_vector[rs->onames[i]]     = CVLINK_NULL;
              }
           }
           else
           {
              link = create_vector[ rs->onames[i]];
              if (link.rs && (link.rs == rs && link.odep_num == i) )
              {
                 create_vector_xt[rs->onames[i]]  = cycle;
                 create_vector_rt[rs->onames[i]]  = sim_cycle;
                 create_vector[rs->onames[i]]     = CVLINK_NULL;
              }
           }
           rs->odep_list[i]= NULL;
        }
     }
  }
}

#define STORE_HASH_SIZE     32
struct spec_mem_ent
{
   md_addr_t  addr;
   unsigned int  data[2];
   struct spec_mem_ent   *next;
};
static struct spec_mem_ent    *bucket_free_list = NULL;
static struct spec_mem_ent    *store_htable[STORE_HASH_SIZE];
static int    fetch_tail;
static int    fetch_head;
static int    fetch_num;
static md_addr_t    pred_PC;
static md_addr_t    recover_PC;
static md_addr_t    fetch_regs_PC;
static md_addr_t    fetch_pred_PC;
static struct fetch_rec    *fetch_data;
```

Figure B-13 Align 2 - Snippet 2

```
static void    cv_init(void);
static void    lsq_init(void);
static void    fetch_init(void);
static void    tracer_init(void);
static void    eventq_init(void);
static void    readyq_init(void);

static tick_t move_event(struct RUU_station *rs);
{
    struct RS_link    *ev;
    struct RS_link    *prev;
    ev   =  event_queue;
    tick_t when    =  0;
    while(ev)
    {
        if(ev->rs == rs)    return  ev->x.when;
        ev   =  ev->next;
    }
    return  when;
}

void sim_init(void)
{
    mem   =  mem_create("mem");
    sim_num_refs  =  0;
    regs_init(regs);
    mem_init(mem);
}

static char    * simoo_reg_obj(struct regs_t *regs, int
               is_write,enum md_reg_type rt,
               int reg, struct eval_value_t *val);
static char    * simoo_mem_obj(struct mem_t *mem,
               int is_write,md_addr_t addr,
               char *p,int nbytes);
static char    * simoo_mstate_obj(FILE *stream, char *cmd,
               regs_t *regs, struct mem_t *mem);
#define        MAX_RS_LINKS    4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
    ld_load_prog(fname, argc, argv, envp, ®s, mem, TRUE);
    if (ptrace_nelt == 2)    ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0)    fatal("bad pipetrace args, use: ");
    fu_pool   =   res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    cv_init();
    ruu_init();
    lsq_init();
    fetch_init();
    tracer_init();
    eventq_init();
    readyq_init();
    rslink_init(MAX_RS_LINKS);
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Figure B-14   Align 3 - Snippet 3

```
typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;

#define MAX_IDEPS 3
#define MAX_ODEPS 2
#define OPERANDS_READY(RS)  \
        (RS)->idep_ready[0] &&     \
        (RS)->idep_ready[1] &&     \
        (RS)->idep_ready[2])

static int RUU_head;
static int RUU_tail;
static int RUU_num;

struct RUU_station  {
   md_inst_t IR;
   enum md_opcode op;
   md_addr_t PC;
   md_addr_tnext_PC;
   md_addr_tpred_PC;
   int ea_comp;
   int in_LSQ;
   int recover_inst;
   int stack_recover_idx;
   struct bpred_update_t dir_update;
   int spec_mode;
   md_addr_t addr;
   INST_TAG_TYPE tag;
   INST_SEQ_TYPE seq;
   unsigned int ptrace_seq;
   int slip;
   int queued;
   int issued;
   int completed;
   int onames[MAX_ODEPS];
   struct RS_link *odep_list[MAX_ODEPS];
   int idep_ready[MAX_IDEPS];
};
static struct RUU_station *RUU;


void sim_uninit(void)  {
   if (ptrace_nelt > 0) ptrace_close();
}

static void ruu_init(void)  {
   RUU = calloc(RUU_size, sizeof(struct RUU_station));
   if (!RUU) fatal("out of virtual memory");
   RUU_num = 0;
   RUU_head = 0;
   RUU_tail = 0;
   RUU_count = 0;
   RUU_fcount = 0;
}
```

Figure B-15   Group 1 - Snippet 1

```
#define STORE_HASH_SIZE 32
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
static md_addr_t fetch_pred_PC;
static struct fetch_rec *fetch_data;
static int fetch_num;
static int fetch_tail, fetch_head;

struct spec_mem_ent & {
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;


static void ruu_writeback(void) {
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event())) {
      rs->completed = TRUE;
      if (rs->recover_inst)  {
        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
          rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++) {
        if ( rs->onames[i] != NA ) {
          if ( rs->spec_mode ) {
            link = spec_create_vector[rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i)) {
              spec_create_vector[rs->onames[i]] = CVLINK_NULL;
              spec_create_vector_rt[rs->onames[i]] = sim_cycle;
              spec_create_vector_xt[rs->onames[i]] = cycle;
            }
          }
          else {
            link = create_vector[ rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i) )  {
              create_vector[ rs->onames[i]]=CVLINK_NULL;
              create_vector_rt[ rs->onames[i]]=sim_cycle;
              create_vector_xt[rs->onames[i]] = cycle;
            }
          }
          rs->odep_list[i]= NULL;
        }
      }
    }
}
```

Figure B-16   Group 2 - Snippet 2

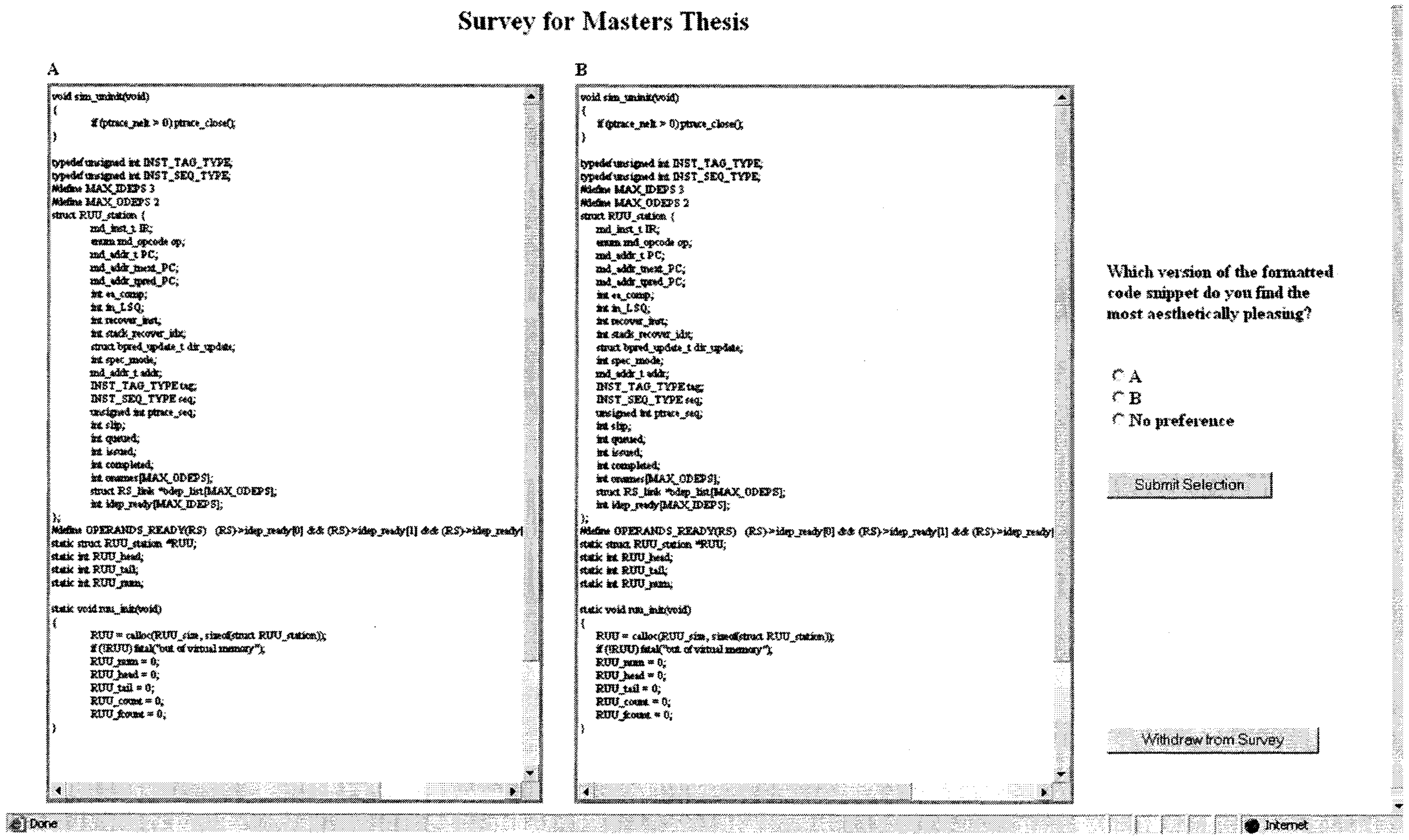
```
#define MAX_RS_LINKS   4096
static void   lsq_init(void);
static void   eventq_init(void);
static void   readyq_init(void);
static void   cv_init(void);
static void   tracer_init(void);
static void   fetch_init(void);
static char  * simoo_reg_obj(struct regs_t *regs,
               int is_write,enum md_reg_type rt,
               int reg, struct eval_value_t *val);
static char   * simoo_mem_obj(struct mem_t *mem,
               int is_write,md_addr_t addr,
               char *p,int nbytes);
static char  * simoo_mstate_obj(FILE *stream, char *cmd,
               regs_t *regs, struct mem_t *mem);


void sim_init(void) {
    sim_num_refs = 0;
    regs_init(®s);
    mem = mem_create("mem");
    mem_init(mem);
}

static tick_t move_event(struct RUU_station *rs)  {
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev) {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_load_prog(char *fname, int argc, char **argv, char **envp) {
    ld_load_prog(fname, argc, argv, envp, ®s, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Figure B-17   Group 3 - Snippet 3

APPENDIX C: CODE COMPARISON CASES

Figure	Case	Snippet
C-1	Case 1 - Indent	Snippet 1
C-2	Case 2 - Indent	Snippet 2
C-3	Case 3 - Indent	Snippet 3
C-4	Case 4 - Margin	Snippet 1
C-5	Case 5 - Margin	Snippet 2
C-6	Case 6 - Margin	Snippet 3
C-7	Case 7 - Align	Snippet 1
C-8	Case 8 - Align	Snippet 2
C-9	Case 9 - Align	Snippet 3
C-10	Case 10 - Group	Snippet 1
C-11	Case 11 - Group	Snippet 2
C-12	Case 12 - Group	Snippet 3
C-13	Case 13 - Indent	Snippet 4

Table C-1 Code Comparison Case in Appendix C



Survey for Masters Thesis

A

```
void sim_uninit(void)
{
        if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
        md_inst_t IR;
        enum md_opcode op;
        md_addr_t PC;
        md_addr_next_PC;
        md_addr_pred_PC;
        int ea_comp;
        int in_LSQ;
        int recover_inst;
        int stack_recover_idx;
        struct bpred_update_t dir_update;
        int spec_mode;
        md_addr_t addr;
        INST_TAG_TYPE tag;
        INST_SEQ_TYPE seq;
        unsigned int ptrace_seq;
        int slip;
        int queued;
        int issued;
        int completed;
        int onames[MAX_ODEPS];
        struct RS_link *odep_list[MAX_ODEPS];
        int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
        RUU = calloc(RUU_size, sizeof(struct RUU_station));
        if (!RUU) fatal("out of virtual memory");
        RUU_num = 0;
        RUU_head = 0;
        RUU_tail = 0;
        RUU_count = 0;
        RUU_fcount = 0;
}
```

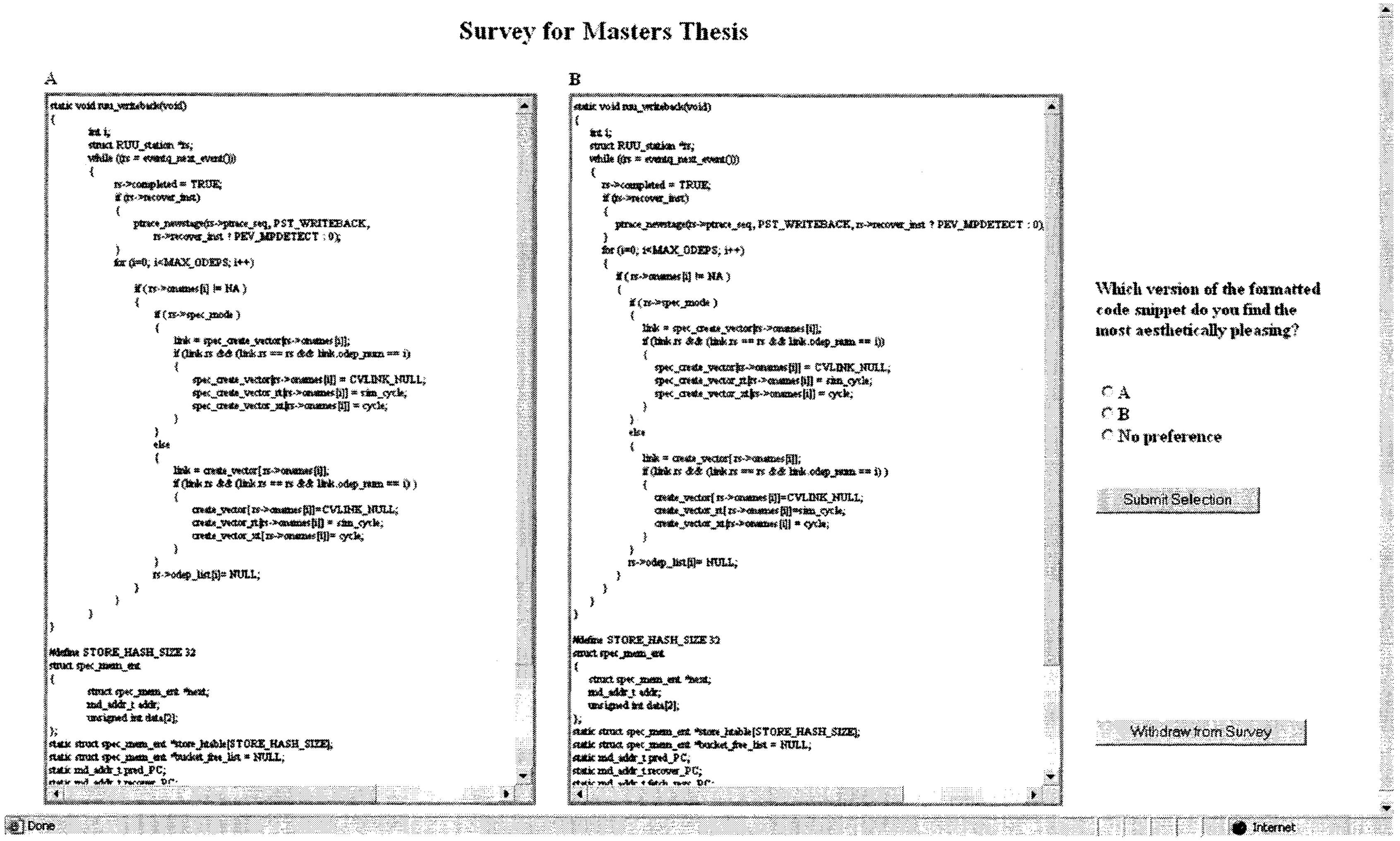
B

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_next_PC;
    md_addr_pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

- ◯ A
- ◯ B
- ◯ No preference

Figure C-1 Case 1 Indent - Snippet 1



Survey for Masters Thesis

A

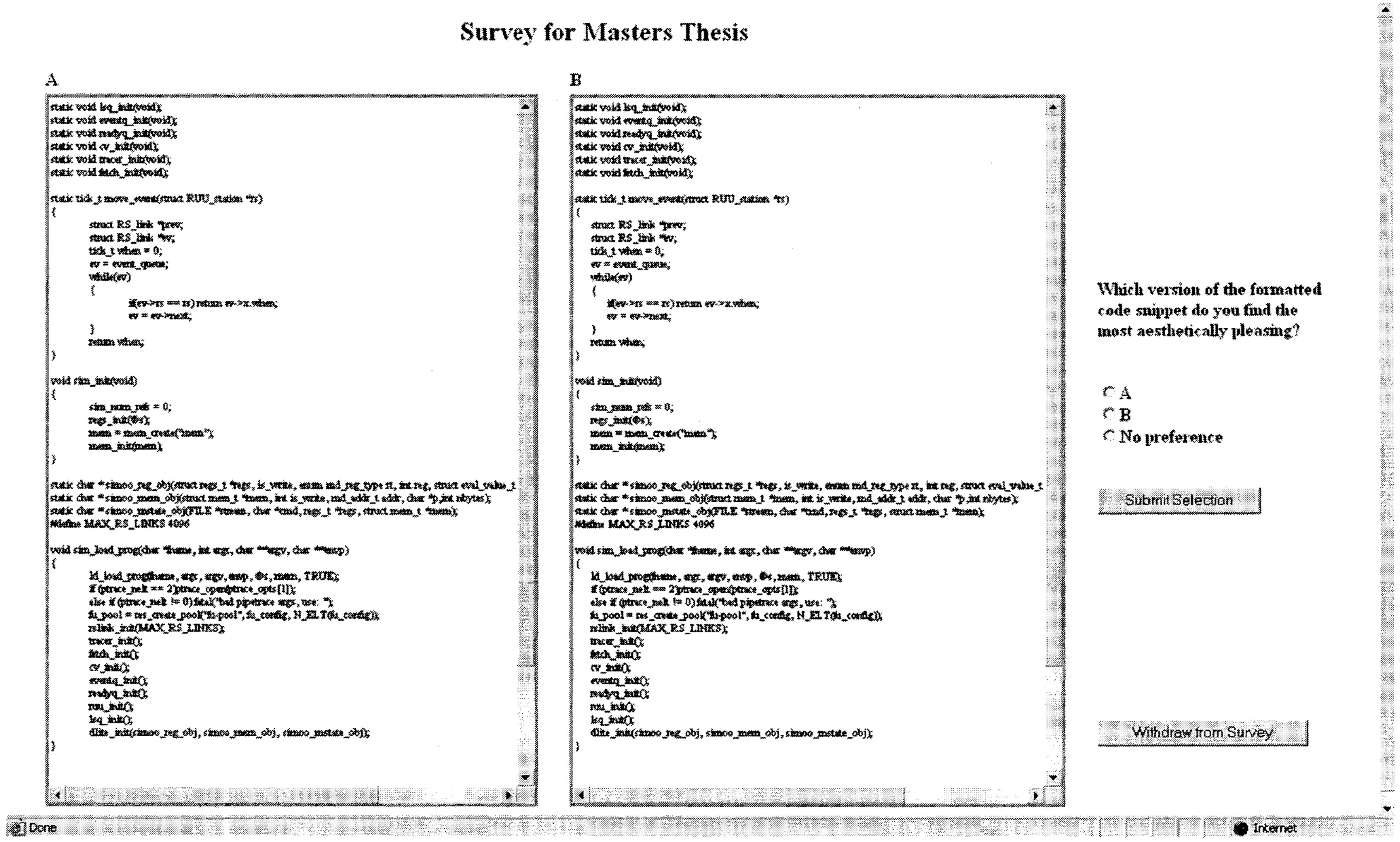
```
static void ruu_writeback(void)
{
        int i;
        struct RUU_station *rs;
        while ((rs = eventq_next_event()))
        {
            rs->completed = TRUE;
            if (rs->recover_inst)
            {
                ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
                    rs->recover_inst ? PEV_MPDETECT : 0);
            }
            for (i=0; i<MAX_ODEPS; i++)

                if ( rs->onames[i] != NA )
                {
                    if ( rs->spec_mode )
                    {
                        link = spec_create_vector[rs->onames[i]];
                        if (link.rs && (link.rs == rs && link.odep_num == i))
                        {
                            spec_create_vector[rs->onames[i]] = CVLINK_NULL;
                            spec_create_vector_rt[rs->onames[i]] = sim_cycle;
                            spec_create_vector_rt[rs->onames[i]] = cycle;
                        }
                    }
                    else
                    {
                        link = create_vector[ rs->onames[i]];
                        if (link.rs && (link.rs == rs && link.odep_num == i) )
                        {
                            create_vector[ rs->onames[i]]=CVLINK_NULL;
                            create_vector_rt[rs->onames[i]] = sim_cycle;
                            create_vector_rt[ rs->onames[i]]= cycle;
                        }
                    }
                    rs->odep_list[i]= NULL;
                }
            }
        }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
        struct spec_mem_ent *next;
        md_addr_t addr;
        unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
```

B

```
static void ruu_writeback(void)
{
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event()))
    {
      rs->completed = TRUE;
      if (rs->recover_inst)
      {
        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
        if ( rs->onames[i] != NA )
        {
          if ( rs->spec_mode )
          {
            link = spec_create_vector[rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i))
            {
              spec_create_vector[rs->onames[i]] = CVLINK_NULL;
              spec_create_vector_rt[rs->onames[i]] = sim_cycle;
              spec_create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          else
          {
            link = create_vector[ rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i) )
            {
              create_vector[ rs->onames[i]]=CVLINK_NULL;
              create_vector_rt[ rs->onames[i]]=sim_cycle;
              create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          rs->odep_list[i]= NULL;
        }
      }
    }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_pred_PC;
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

Figure C-2 Case 2 Indent - Snippet 2



Survey for Masters Thesis

A

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
        struct RS_link *prev;
        struct RS_link *ev;
        tick_t when = 0;
        ev = event_queue;
        while(ev)
        {
                if(ev->rs == rs) return ev->x.when;
                ev = ev->next;
        }
        return when;
}

void sim_init(void)
{
        sim_num_refs = 0;
        regs_init(&regs);
        mem = mem_create("mem");
        mem_init(mem);
}

static char *simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t
static char *simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p, int nbytes);
static char *simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
        ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
        if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
        else if (ptrace_nelt != 0)fatal("bad pipetrace args, use: ");
        fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
        rslink_init(MAX_RS_LINKS);
        tracer_init();
        fetch_init();
        cv_init();
        eventq_init();
        readyq_init();
        ruu_init();
        lsq_init();
        dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

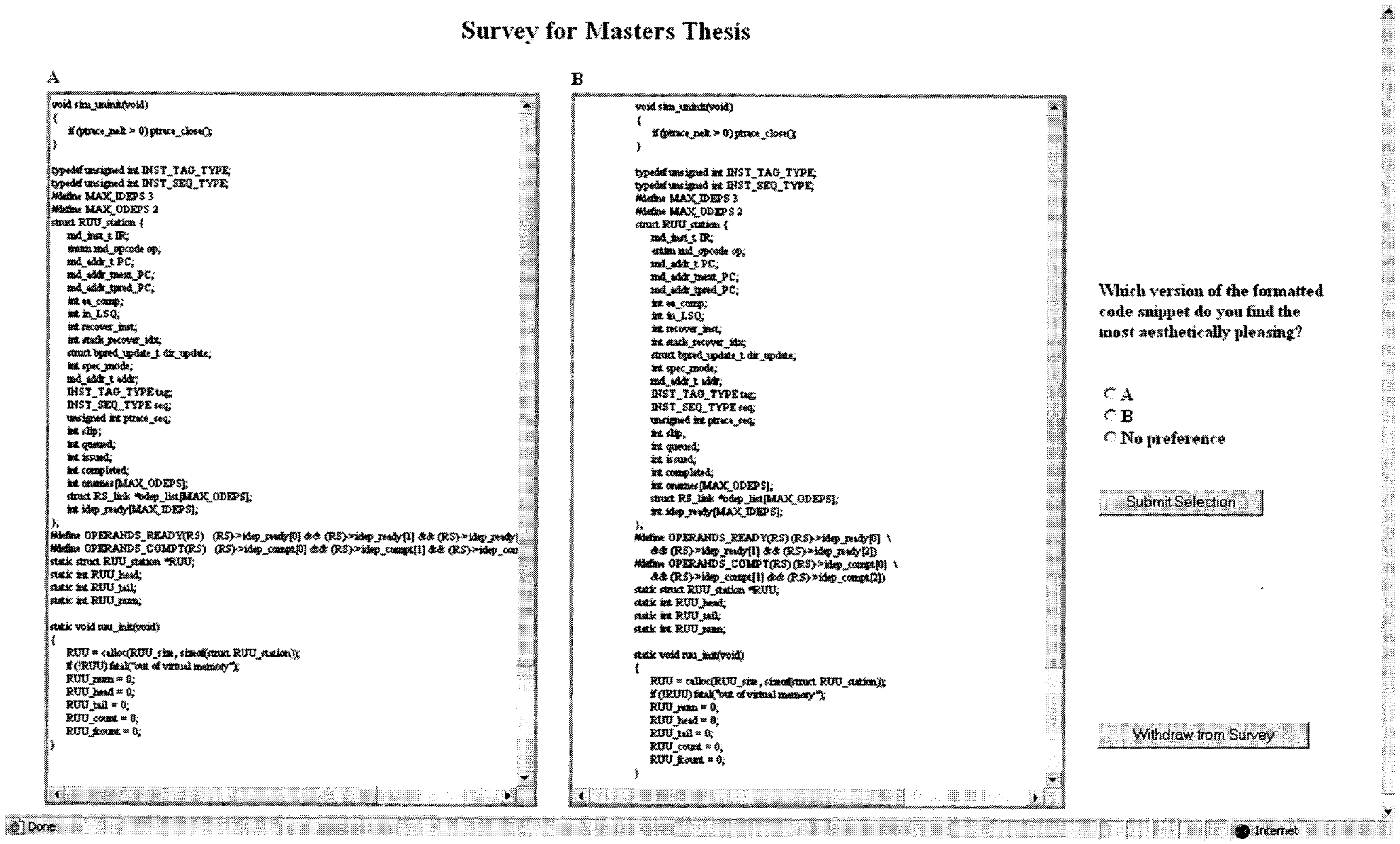
B

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
  struct RS_link *prev;
  struct RS_link *ev;
  tick_t when = 0;
  ev = event_queue;
  while(ev)
  {
    if(ev->rs == rs) return ev->x.when;
    ev = ev->next;
  }
  return when;
}

void sim_init(void)
{
  sim_num_refs = 0;
  regs_init(&regs);
  mem = mem_create("mem");
  mem_init(mem);
}

static char *simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t
static char *simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p, int nbytes);
static char *simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
  ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
  if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
  else if (ptrace_nelt != 0)fatal("bad pipetrace args, use: ");
  fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
  rslink_init(MAX_RS_LINKS);
  tracer_init();
  fetch_init();
  cv_init();
  eventq_init();
  readyq_init();
  ruu_init();
  lsq_init();
  dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-3 Case 3 Indent - Snippet 3



Survey for Masters Thesis

A

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_t next_PC;
    md_addr_t pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready
#define OPERANDS_COMPT(RS)   (RS)->idep_compt[0] && (RS)->idep_compt[1] && (RS)->idep_co
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

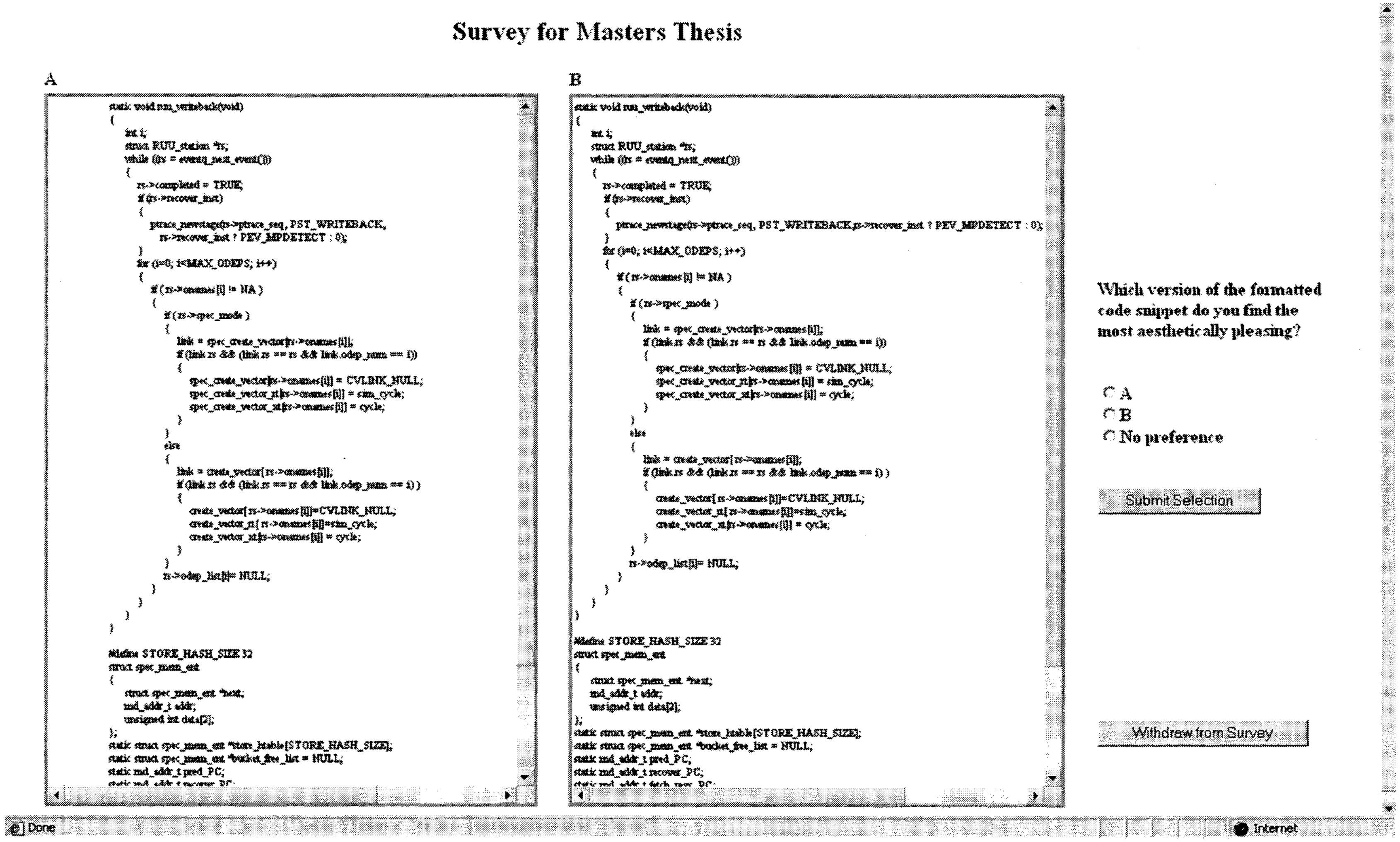
B

```
            void sim_uninit(void)
            {
                if (ptrace_nelt > 0) ptrace_close();
            }

            typedef unsigned int INST_TAG_TYPE;
            typedef unsigned int INST_SEQ_TYPE;
            #define MAX_IDEPS 3
            #define MAX_ODEPS 2
            struct RUU_station {
                md_inst_t IR;
                enum md_opcode op;
                md_addr_t PC;
                md_addr_t next_PC;
                md_addr_t pred_PC;
                int ea_comp;
                int in_LSQ;
                int recover_inst;
                int stack_recover_idx;
                struct bpred_update_t dir_update;
                int spec_mode;
                md_addr_t addr;
                INST_TAG_TYPE tag;
                INST_SEQ_TYPE seq;
                unsigned int ptrace_seq;
                int slip;
                int queued;
                int issued;
                int completed;
                int onames[MAX_ODEPS];
                struct RS_link *odep_list[MAX_ODEPS];
                int idep_ready[MAX_IDEPS];
            };
            #define OPERANDS_READY(RS) (RS)->idep_ready[0] \
                && (RS)->idep_ready[1] && (RS)->idep_ready[2])
            #define OPERANDS_COMPT(RS) (RS)->idep_compt[0] \
                && (RS)->idep_compt[1] && (RS)->idep_compt[2])
            static struct RUU_station *RUU;
            static int RUU_head;
            static int RUU_tail;
            static int RUU_num;

            static void ruu_init(void)
            {
                RUU = calloc(RUU_size, sizeof(struct RUU_station));
                if (!RUU) fatal("out of virtual memory");
                RUU_num = 0;
                RUU_head = 0;
                RUU_tail = 0;
                RUU_count = 0;
                RUU_fcount = 0;
            }
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

- ◯ A
- ◯ B
- ◯ No preference

Figure C-4 Case 4 Margin - Snippet 1



Survey for Masters Thesis

A

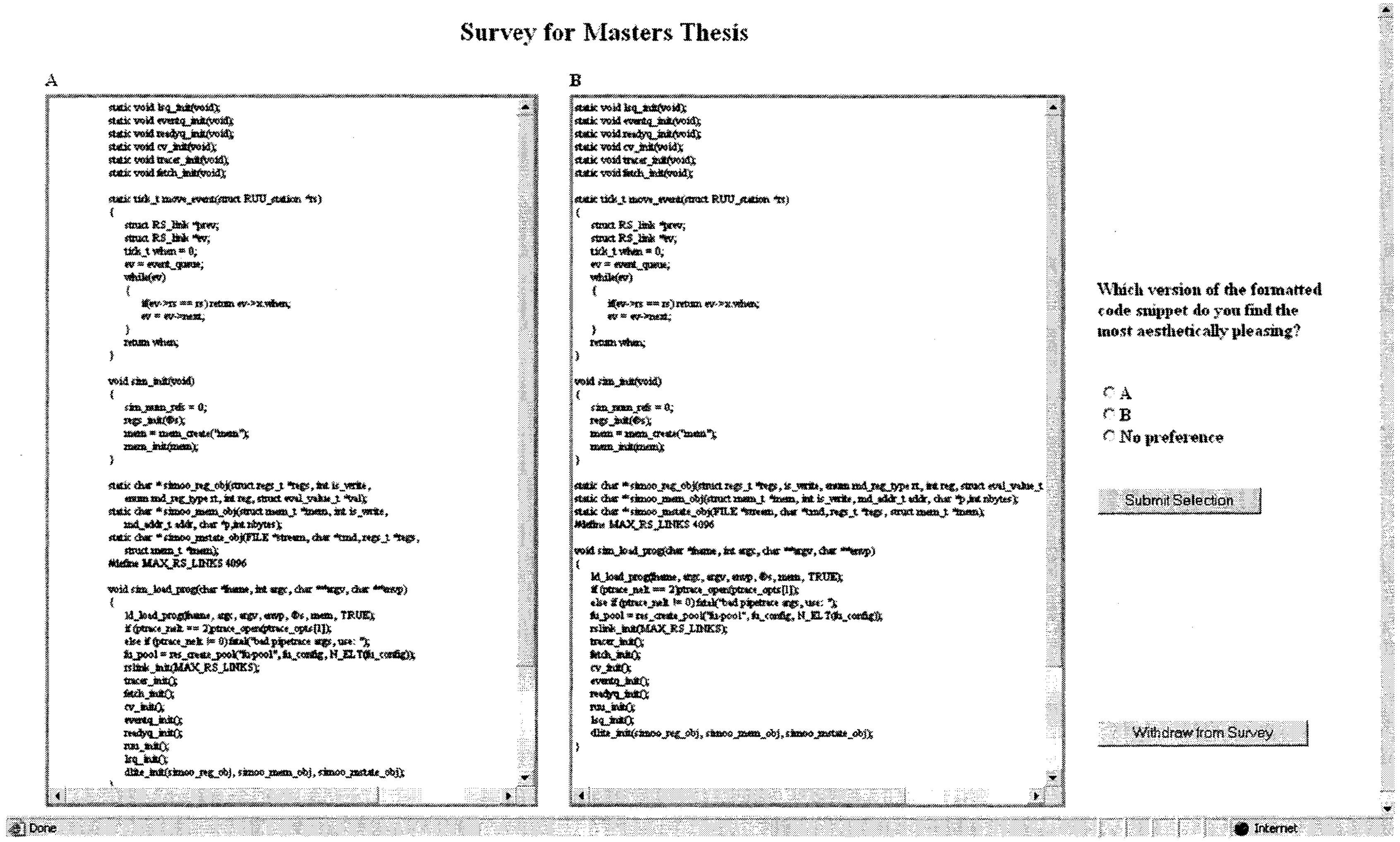
```
static void ruu_writeback(void)
{
   int i;
   struct RUU_station *rs;
   while ((rs = eventq_next_event()))
   {
      rs->completed = TRUE;
      if (rs->recover_inst)
      {
         ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
            rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
         if (rs->onames[i] != NA )
         {
            if (rs->spec_mode )
            {
               link = spec_create_vector[rs->onames[i]];
               if (link.rs && (link.rs == rs && link.odep_num == i))
               {
                  spec_create_vector[rs->onames[i]] = CVLINK_NULL;
                  spec_create_vector_rt[rs->onames[i]] = sim_cycle;
                  spec_create_vector_xt[rs->onames[i]] = cycle;
               }
            }
            else
            {
               link = create_vector[rs->onames[i]];
               if (link.rs && (link.rs == rs && link.odep_num == i) )
               {
                  create_vector[rs->onames[i]]=CVLINK_NULL;
                  create_vector_rt[rs->onames[i]]=sim_cycle;
                  create_vector_xt[rs->onames[i]] = cycle;
               }
            }
            rs->odep_list[i]= NULL;
         }
      }
   }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
   struct spec_mem_ent *next;
   md_addr_t addr;
   unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
```

B

```
static void ruu_writeback(void)
{
  int i;
  struct RUU_station *rs;
  while ((rs = eventq_next_event()))
  {
    rs->completed = TRUE;
    if (rs->recover_inst)
    {
      ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,rs->recover_inst ? PEV_MPDETECT : 0);
    }
    for (i=0; i<MAX_ODEPS; i++)
    {
      if (rs->onames[i] != NA )
      {
        if (rs->spec_mode )
        {
          link = spec_create_vector[rs->onames[i]];
          if (link.rs && (link.rs == rs && link.odep_num == i))
          {
            spec_create_vector[rs->onames[i]] = CVLINK_NULL;
            spec_create_vector_rt[rs->onames[i]] = sim_cycle;
            spec_create_vector_xt[rs->onames[i]] = cycle;
          }
        }
        else
        {
          link = create_vector[rs->onames[i]];
          if (link.rs && (link.rs == rs && link.odep_num == i) )
          {
            create_vector[rs->onames[i]]=CVLINK_NULL;
            create_vector_rt[rs->onames[i]]=sim_cycle;
            create_vector_xt[rs->onames[i]] = cycle;
          }
        }
        rs->odep_list[i]= NULL;
      }
    }
  }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
  struct spec_mem_ent *next;
  md_addr_t addr;
  unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

Figure C-5 Case 5 Margin - Snippet 2



Survey for Masters Thesis

A

```
            static void lsq_init(void);
            static void eventq_init(void);
            static void readyq_init(void);
            static void cv_init(void);
            static void tracer_init(void);
            static void fetch_init(void);

            static tick_t move_event(struct RUU_station *rs)
            {
                struct RS_link *prev;
                struct RS_link *ev;
                tick_t when = 0;
                ev = event_queue;
                while(ev)
                {
                    if(ev->rs == rs) return ev->x.when;
                    ev = ev->next;
                }
                return when;
            }

            void sim_init(void)
            {
                sim_num_refs = 0;
                regs_init(&regs);
                mem = mem_create("mem");
                mem_init(mem);
            }

            static char * simoo_reg_obj(struct regs_t *regs, int is_write,
                enum md_reg_type rt, int reg, struct eval_value_t *val);
            static char * simoo_mem_obj(struct mem_t *mem, int is_write,
                md_addr_t addr, char *p,int nbytes);
            static char * simoo_mstate_obj(FILE *stream, char *cmd,regs_t *regs,
                struct mem_t *mem);
            #define MAX_RS_LINKS 4096

            void sim_load_prog(char *fname, int argc, char **argv, char **envp)
            {
                ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
                if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
                else if (ptrace_nelt != 0)fatal("bad pipetrace args, use: ");
                fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
                rslink_init(MAX_RS_LINKS);
                tracer_init();
                fetch_init();
                cv_init();
                eventq_init();
                readyq_init();
                ruu_init();
                lsq_init();
                dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
```

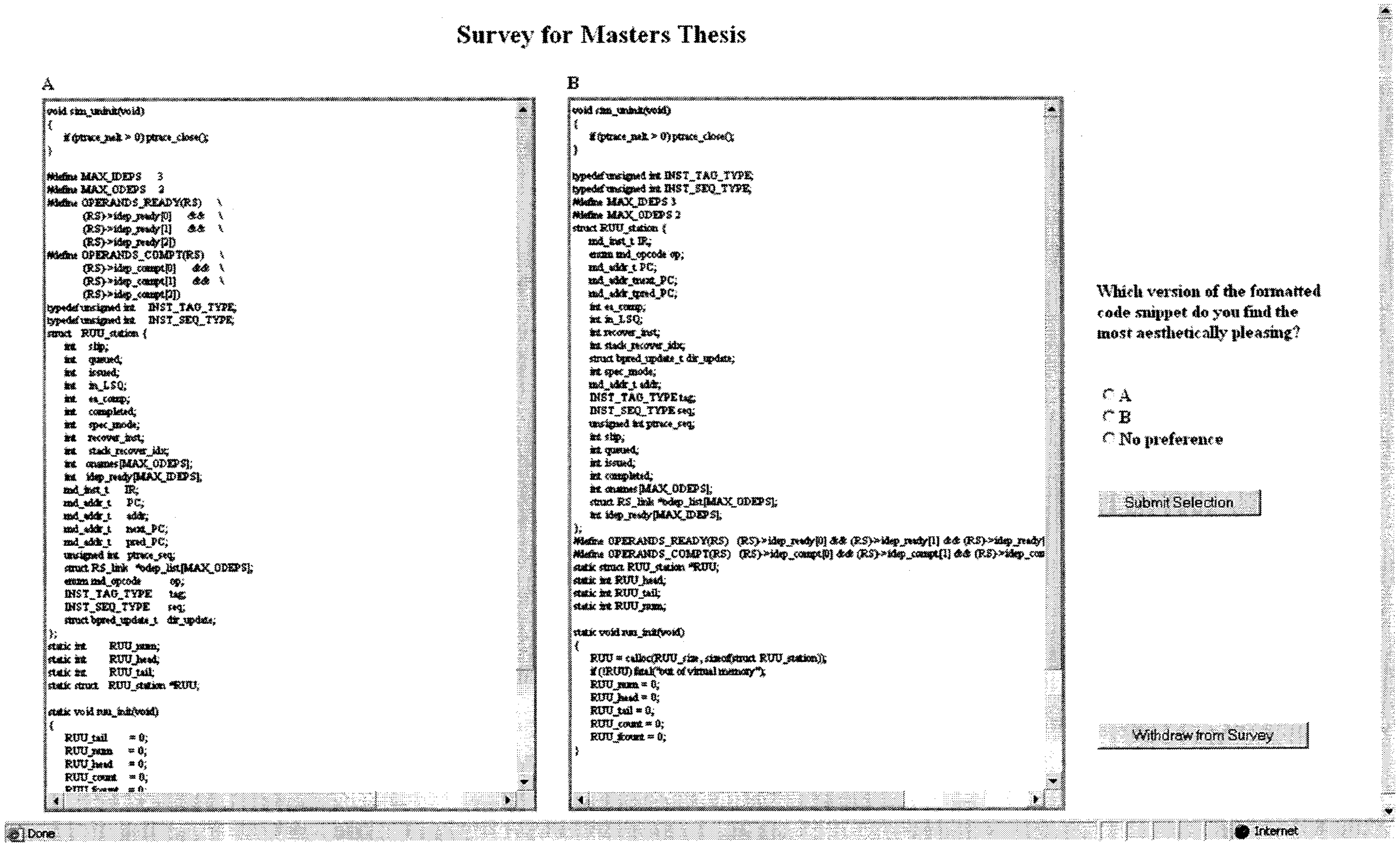
B

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev)
    {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_init(void)
{
    sim_num_refs = 0;
    regs_init(&regs);
    mem = mem_create("mem");
    mem_init(mem);
}

static char *simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t
static char *simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p,int nbytes);
static char *simoo_mstate_obj(FILE *stream, char *cmd,regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
    ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0)fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

- ○ A
- ○ B
- ○ No preference

Figure C-6 Case 6 Margin - Snippet 3


Survey for Masters Thesis

A

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

#define MAX_IDEPS     3
#define MAX_ODEPS     2
#define OPERANDS_READY(RS)      \
        (RS)->idep_ready[0]   &&  \
        (RS)->idep_ready[1]   &&  \
        (RS)->idep_ready[2])
#define OPERANDS_COMPT(RS)      \
        (RS)->idep_compt[0]   &&  \
        (RS)->idep_compt[1]   &&  \
        (RS)->idep_compt[2])
typedef unsigned int    INST_TAG_TYPE;
typedef unsigned int    INST_SEQ_TYPE;
struct  RUU_station {
    int    slip;
    int    queued;
    int    issued;
    int    in_LSQ;
    int    ea_comp;
    int    completed;
    int    spec_mode;
    int    recover_inst;
    int    stack_recover_idx;
    int    onames[MAX_ODEPS];
    int    idep_ready[MAX_IDEPS];
    md_inst_t     IR;
    md_addr_t     PC;
    md_addr_t     addr;
    md_addr_t     next_PC;
    md_addr_t     pred_PC;
    unsigned int   ptrace_seq;
    struct RS_link   *odep_list[MAX_ODEPS];
    enum md_opcode          op;
    INST_TAG_TYPE        tag;
    INST_SEQ_TYPE        seq;
    struct bpred_update_t   dir_update;
};
static int       RUU_num;
static int       RUU_head;
static int       RUU_tail;
static struct    RUU_station *RUU;

static void ruu_init(void)
{
    RUU_tail      = 0;
    RUU_num       = 0;
    RUU_head      = 0;
    RUU_count     = 0;
    RUU_fcount    = 0;
```

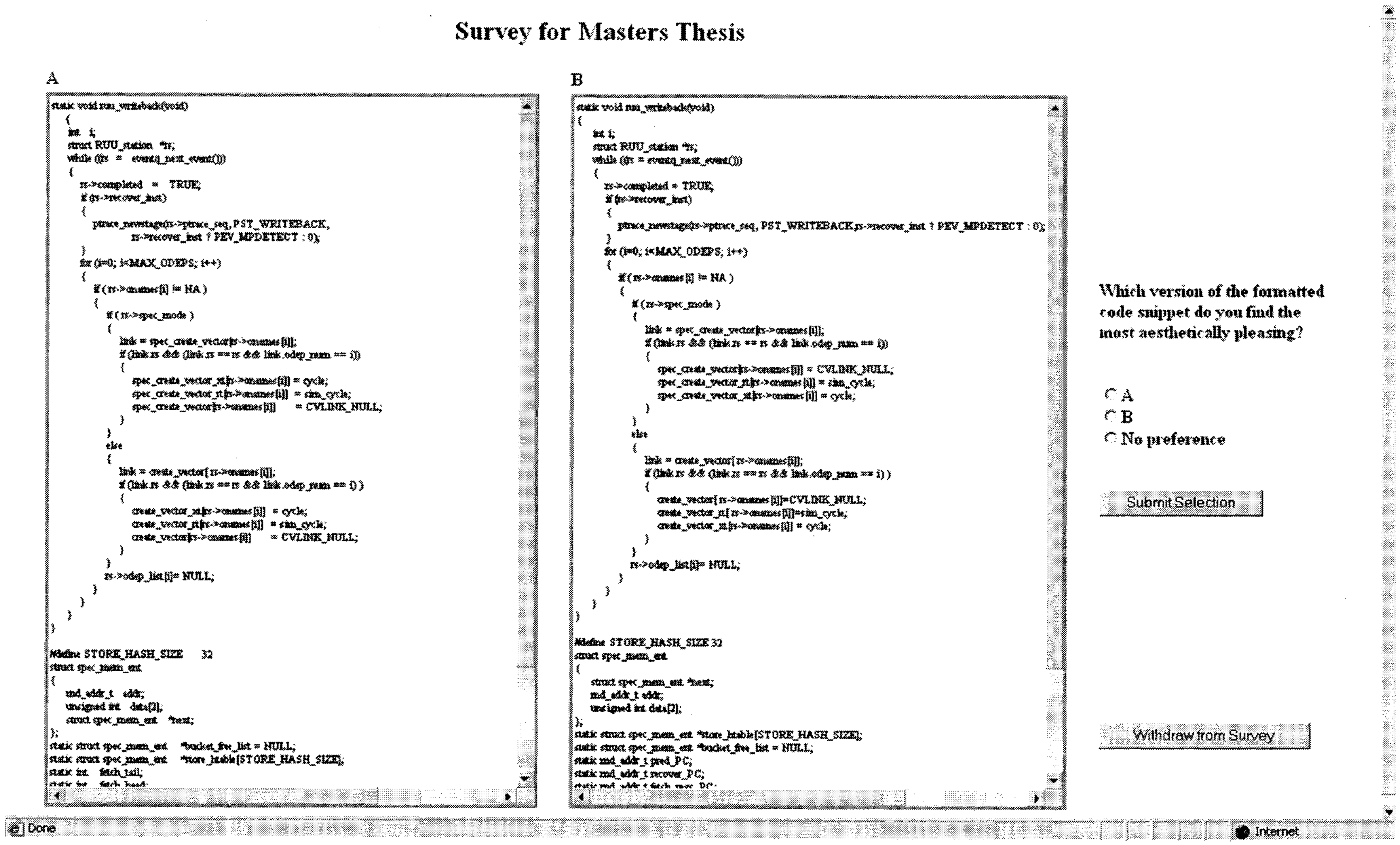
B

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_t next_PC;
    md_addr_t pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[
#define OPERANDS_COMPT(RS)   (RS)->idep_compt[0] && (RS)->idep_compt[1] && (RS)->idep_com
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-7   Case 7   Align - Snippet 1



Survey for Masters Thesis

A

```
static void ruu_writeback(void)
   {
   int   i;
   struct RUU_station  *rs;
   while ((rs  =   eventq_next_event()))
   {
      rs->completed   =   TRUE;
      if (rs->recover_inst)
      {
         ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
                  rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
         if ( rs->onames[i] != NA )
         {
            if ( rs->spec_mode )
            {
               link = spec_create_vector[rs->onames[i]];
               if (link.rs && (link.rs == rs && link.odep_num == i))
               {
                  spec_create_vector_rt[rs->onames[i]] = cycle;
                  spec_create_vector_rt[rs->onames[i]]  = sim_cycle;
                  spec_create_vector[rs->onames[i]]       = CVLINK_NULL;
               }
            }
            else
            {
               link = create_vector[ rs->onames[i]];
               if (link.rs  && (link.rs == rs  && link.odep_num == i) )
               {
                  create_vector_rt[rs->onames[i]]  = cycle;
                  create_vector_rt[rs->onames[i]]  = sim_cycle;
                  create_vector[rs->onames[i]]       = CVLINK_NULL;
               }
            }
            rs->odep_list[i]= NULL;
         }
      }
   }
}

#define STORE_HASH_SIZE      32
struct spec_mem_ent
{
   md_addr_t   addr;
   unsigned int   data[2];
   struct spec_mem_ent   *next;
};
static struct spec_mem_ent    *bucket_free_list = NULL;
static struct spec_mem_ent    *store_htable[STORE_HASH_SIZE];
static int   fetch_tail;
static int   fetch_head;
```

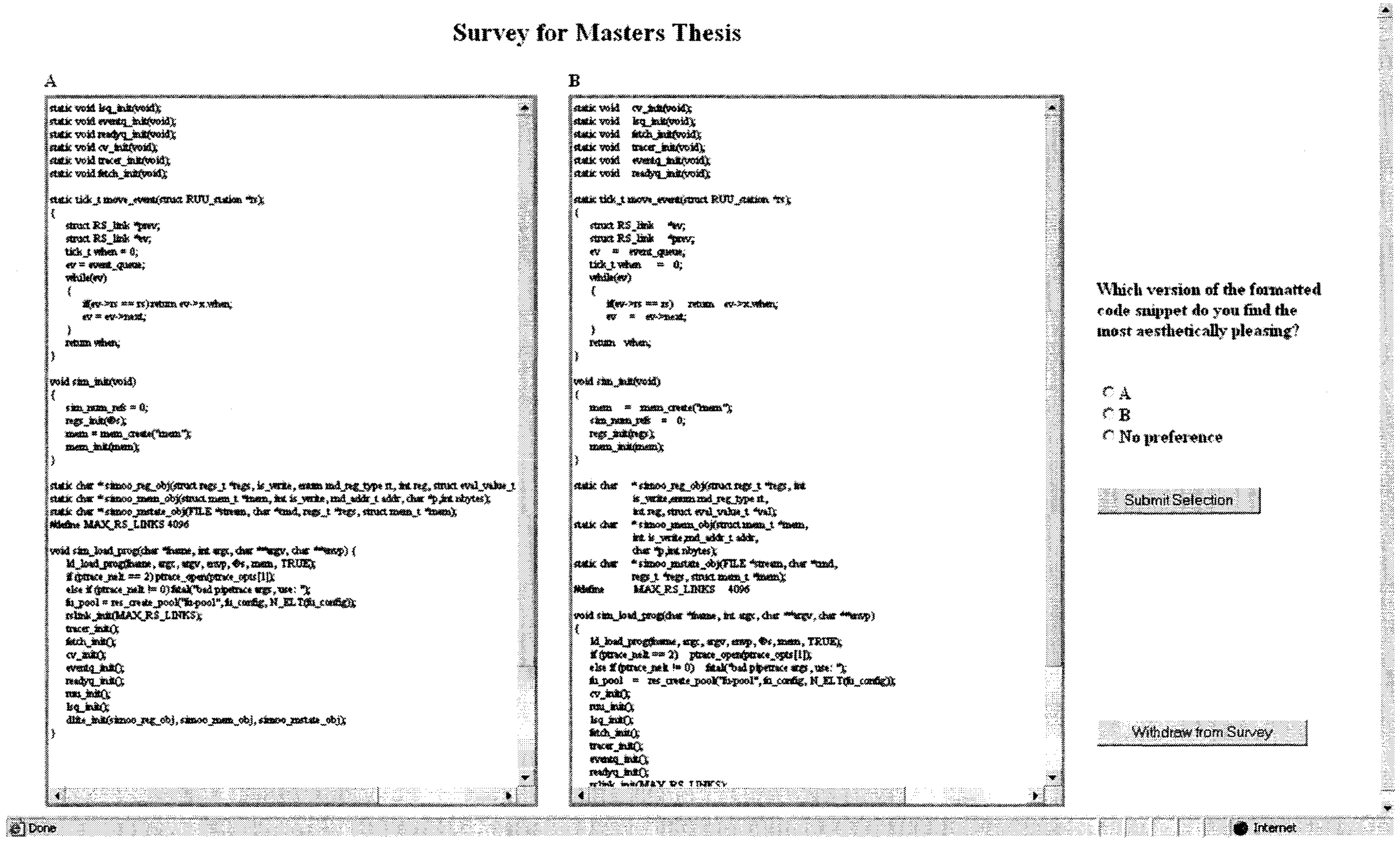
B

```
static void ruu_writeback(void)
{
   int i;
   struct RUU_station *rs;
   while ((rs = eventq_next_event()))
   {
      rs->completed = TRUE;
      if (rs->recover_inst)
      {
         ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
         if ( rs->onames[i] != NA )
         {
            if ( rs->spec_mode )
            {
               link = spec_create_vector[rs->onames[i]];
               if (link.rs && (link.rs == rs && link.odep_num == i))
               {
                  spec_create_vector[rs->onames[i]] = CVLINK_NULL;
                  spec_create_vector_rt[rs->onames[i]] = sim_cycle;
                  spec_create_vector_rt[rs->onames[i]] = cycle;
               }
            }
            else
            {
               link = create_vector[ rs->onames[i]];
               if (link.rs && (link.rs == rs && link.odep_num == i) )
               {
                  create_vector[ rs->onames[i]]=CVLINK_NULL;
                  create_vector_rt[ rs->onames[i]]=sim_cycle;
                  create_vector_rt[rs->onames[i]] = cycle;
               }
            }
            rs->odep_list[i]= NULL;
         }
      }
   }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
   struct spec_mem_ent *next;
   md_addr_t addr;
   unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-8   Case 8   Align - Snippet 2



Survey for Masters Thesis

A

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs);
{
   struct RS_link *prev;
   struct RS_link *ev;
   tick_t when = 0;
   ev = event_queue;
   while(ev)
   {
      if(ev->rs == rs)return ev->x.when;
      ev = ev->next;
   }
   return when;
}

void sim_init(void)
{
   sim_num_refs = 0;
   regs_init(&regs);
   mem = mem_create("mem");
   mem_init(mem);
}

static char *simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t
static char *simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p,int nbytes);
static char *simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs, struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp) {
   ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
   if (ptrace_nelt == 2) ptrace_open(ptrace_opts[1]);
   else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
   fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
   rslink_init(MAX_RS_LINKS);
   tracer_init();
   fetch_init();
   cv_init();
   eventq_init();
   readyq_init();
   ruu_init();
   lsq_init();
   dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

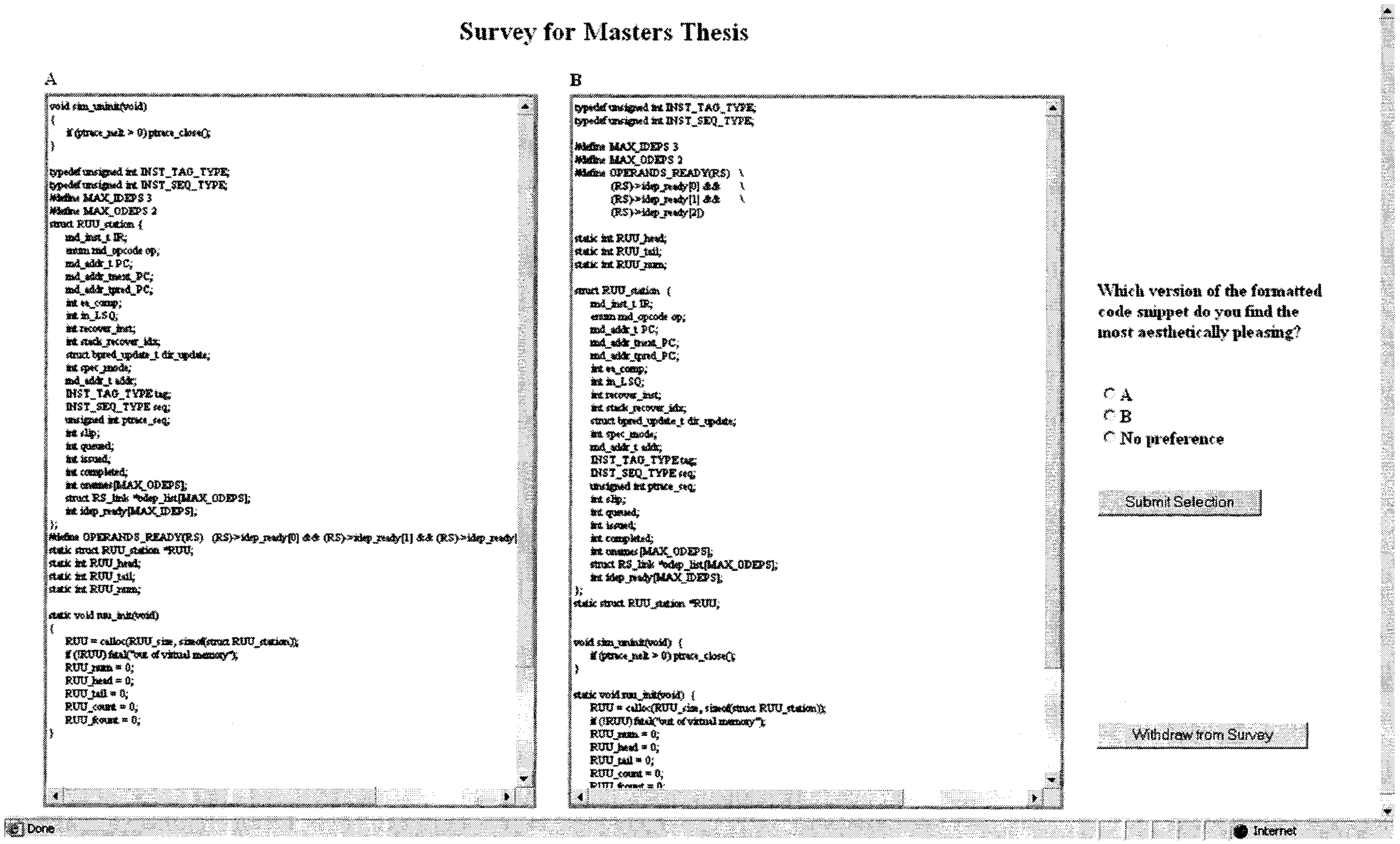
B

```
static void    cv_init(void);
static void    lsq_init(void);
static void    fetch_init(void);
static void    tracer_init(void);
static void    eventq_init(void);
static void    readyq_init(void);

static tick_t move_event(struct RUU_station *rs);
{
   struct RS_link    *ev;
   struct RS_link    *prev;
   ev    =   event_queue;
   tick_t when     =   0;
   while(ev)
   {
      if(ev->rs == rs)    return   ev->x.when;
      ev    =    ev->next;
   }
   return   when;
}

void sim_init(void)
{
   mem     =   mem_create("mem");
   sim_num_refs   =   0;
   regs_init(regs);
   mem_init(mem);
}

static char    *simoo_reg_obj(struct regs_t *regs, int
               is_write,enum md_reg_type rt,
               int reg, struct eval_value_t *val);
static char    *simoo_mem_obj(struct mem_t *mem,
               int is_write,md_addr_t addr,
               char *p,int nbytes);
static char    *simoo_mstate_obj(FILE *stream, char *cmd,
               regs_t *regs, struct mem_t *mem);
#define        MAX_RS_LINKS    4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
   ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
   if (ptrace_nelt == 2)    ptrace_open(ptrace_opts[1]);
   else if (ptrace_nelt != 0)    fatal("bad pipetrace args, use: ");
   fu_pool   =   res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
   cv_init();
   ruu_init();
   lsq_init();
   fetch_init();
   tracer_init();
   eventq_init();
   readyq_init();
   rslink_init(MAX_RS_LINKS);
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-9 Case 9 Align - Snippet 3



Survey for Masters Thesis

A

```
void sim_uninit(void)
{
    if (ptrace_nelt > 0) ptrace_close();
}

typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;
#define MAX_IDEPS 3
#define MAX_ODEPS 2
struct RUU_station {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_t next_PC;
    md_addr_t pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
#define OPERANDS_READY(RS)   (RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)->idep_ready[
static struct RUU_station *RUU;
static int RUU_head;
static int RUU_tail;
static int RUU_num;

static void ruu_init(void)
{
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
}
```

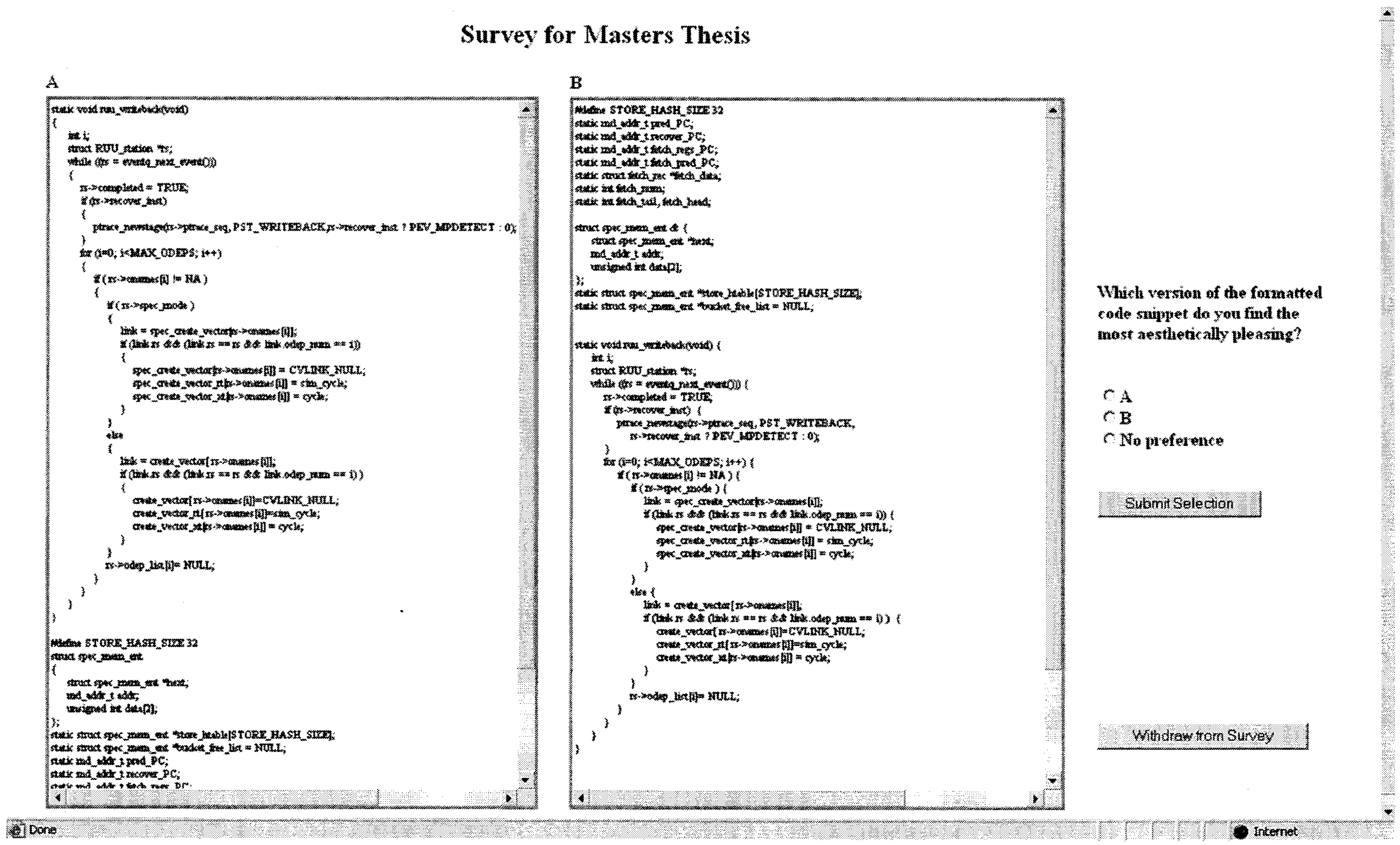
B

```
typedef unsigned int INST_TAG_TYPE;
typedef unsigned int INST_SEQ_TYPE;

#define MAX_IDEPS 3
#define MAX_ODEPS 2
#define OPERANDS_READY(RS)  \
        (RS)->idep_ready[0] &&      \
        (RS)->idep_ready[1] &&      \
        (RS)->idep_ready[2])

static int RUU_head;
static int RUU_tail;
static int RUU_num;

struct RUU_station  {
    md_inst_t IR;
    enum md_opcode op;
    md_addr_t PC;
    md_addr_t next_PC;
    md_addr_t pred_PC;
    int ea_comp;
    int in_LSQ;
    int recover_inst;
    int stack_recover_idx;
    struct bpred_update_t dir_update;
    int spec_mode;
    md_addr_t addr;
    INST_TAG_TYPE tag;
    INST_SEQ_TYPE seq;
    unsigned int ptrace_seq;
    int slip;
    int queued;
    int issued;
    int completed;
    int onames[MAX_ODEPS];
    struct RS_link *odep_list[MAX_ODEPS];
    int idep_ready[MAX_IDEPS];
};
static struct RUU_station *RUU;


void sim_uninit(void)  {
    if (ptrace_nelt > 0) ptrace_close();
}

static void ruu_init(void)  {
    RUU = calloc(RUU_size, sizeof(struct RUU_station));
    if (!RUU) fatal("out of virtual memory");
    RUU_num = 0;
    RUU_head = 0;
    RUU_tail = 0;
    RUU_count = 0;
    RUU_fcount = 0;
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-10 Case 10 Group - Snippet 1


Survey for Masters Thesis

A

```
static void ruu_writeback(void)
{
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event()))
    {
      rs->completed = TRUE;
      if (rs->recover_inst)
      {
        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++)
      {
        if ( rs->onames[i] != NA )
        {
          if ( rs->spec_mode )
          {
            link = spec_create_vector[rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i))
            {
              spec_create_vector[rs->onames[i]] = CVLINK_NULL;
              spec_create_vector_rt[rs->onames[i]] = sim_cycle;
              spec_create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          else
          {
            link = create_vector[ rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i) )
            {
              create_vector[ rs->onames[i]]=CVLINK_NULL;
              create_vector_rt[ rs->onames[i]]=sim_cycle;
              create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          rs->odep_list[i]= NULL;
        }
      }
    }
}

#define STORE_HASH_SIZE 32
struct spec_mem_ent
{
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;
static md_addr_t pred_PC;
static md_addr_t recover_PC;
```

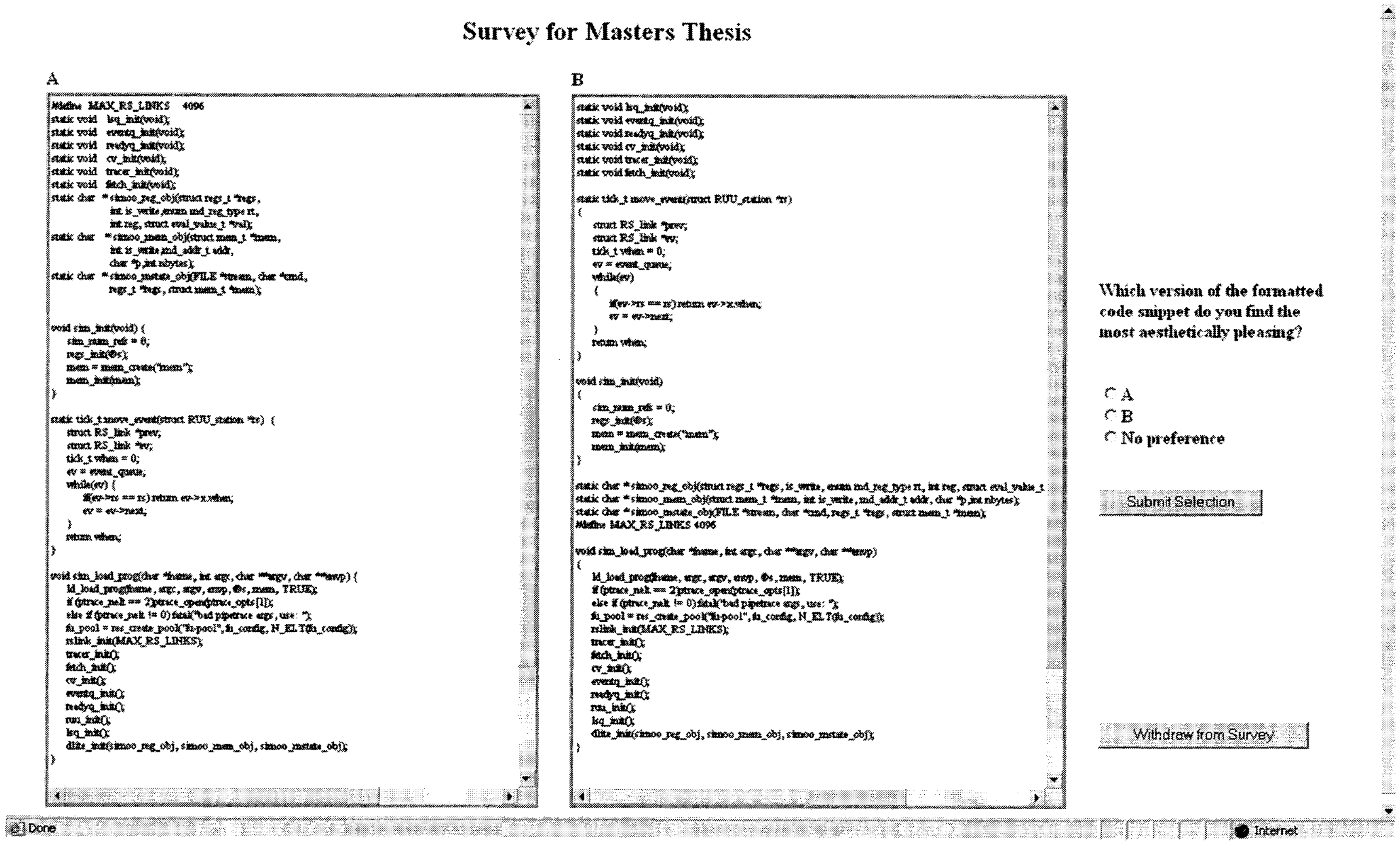
B

```
#define STORE_HASH_SIZE 32
static md_addr_t pred_PC;
static md_addr_t recover_PC;
static md_addr_t fetch_regs_PC;
static md_addr_t fetch_pred_PC;
static struct fetch_rec *fetch_data;
static int fetch_num;
static int fetch_tail, fetch_head;

struct spec_mem_ent & {
    struct spec_mem_ent *next;
    md_addr_t addr;
    unsigned int data[2];
};
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];
static struct spec_mem_ent *bucket_free_list = NULL;


static void ruu_writeback(void) {
    int i;
    struct RUU_station *rs;
    while ((rs = eventq_next_event())) {
      rs->completed = TRUE;
      if (rs->recover_inst) {
        ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
          rs->recover_inst ? PEV_MPDETECT : 0);
      }
      for (i=0; i<MAX_ODEPS; i++) {
        if ( rs->onames[i] != NA ) {
          if ( rs->spec_mode ) {
            link = spec_create_vector[rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i)) {
              spec_create_vector[rs->onames[i]] = CVLINK_NULL;
              spec_create_vector_rt[rs->onames[i]] = sim_cycle;
              spec_create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          else {
            link = create_vector[ rs->onames[i]];
            if (link.rs && (link.rs == rs && link.odep_num == i) ) {
              create_vector[ rs->onames[i]]=CVLINK_NULL;
              create_vector_rt[ rs->onames[i]]=sim_cycle;
              create_vector_rt[rs->onames[i]] = cycle;
            }
          }
          rs->odep_list[i]= NULL;
        }
      }
    }
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

- ○ A
- ○ B
- ○ No preference

Figure C-11   Case 11   Group - Snippet 2



Survey for Masters Thesis

A

```
#define  MAX_RS_LINKS    4096
static void   lsq_init(void);
static void   eventq_init(void);
static void   readyq_init(void);
static void   cv_init(void);
static void   tracer_init(void);
static void   fetch_init(void);
static char  *simoo_reg_obj(struct regs_t *regs,
               int is_write,enum md_reg_type rt,
               int reg, struct eval_value_t *val);
static char  *simoo_mem_obj(struct mem_t *mem,
               int is_write,md_addr_t addr,
               char *p,int nbytes);
static char  *simoo_mstate_obj(FILE *stream, char *cmd,
               regs_t *regs, struct mem_t *mem);


void sim_init(void) {
    sim_num_refs = 0;
    regs_init(&regs);
    mem = mem_create("mem");
    mem_init(mem);
}

static tick_t move_event(struct RUU_station *rs)  {
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev) {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_load_prog(char *fname, int argc, char **argv, char **envp) {
    ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0) fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

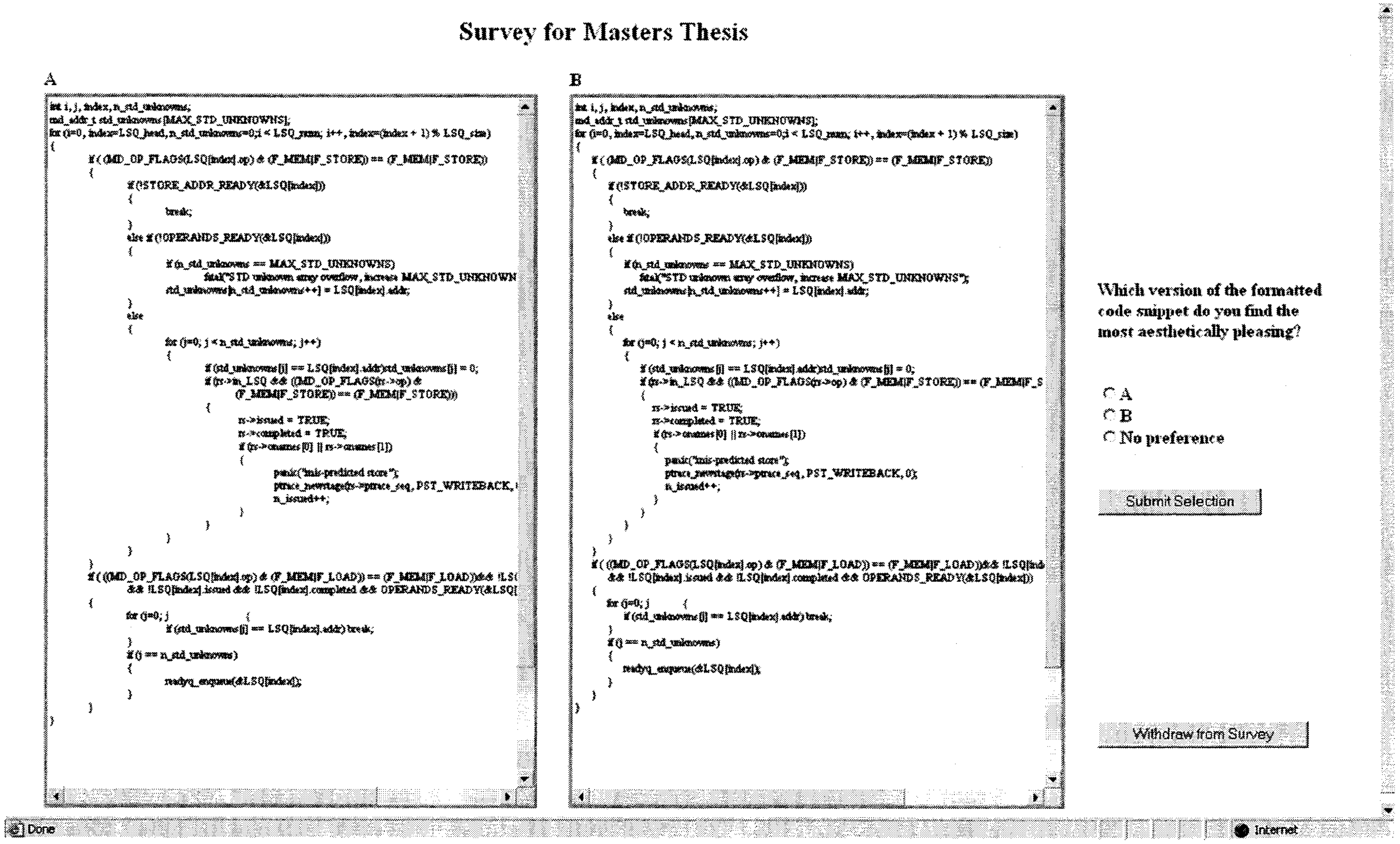
B

```
static void lsq_init(void);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

static tick_t move_event(struct RUU_station *rs)
{
    struct RS_link *prev;
    struct RS_link *ev;
    tick_t when = 0;
    ev = event_queue;
    while(ev)
    {
        if(ev->rs == rs) return ev->x.when;
        ev = ev->next;
    }
    return when;
}

void sim_init(void)
{
    sim_num_refs = 0;
    regs_init(&regs);
    mem = mem_create("mem");
    mem_init(mem);
}

static char *simoo_reg_obj(struct regs_t *regs, is_write, enum md_reg_type rt, int reg, struct eval_value_t
static char *simoo_mem_obj(struct mem_t *mem, int is_write, md_addr_t addr, char *p,int nbytes);
static char *simoo_mstate_obj(FILE *stream, char *cmd, regs_t *regs , struct mem_t *mem);
#define MAX_RS_LINKS 4096

void sim_load_prog(char *fname, int argc, char **argv, char **envp)
{
    ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);
    if (ptrace_nelt == 2)ptrace_open(ptrace_opts[1]);
    else if (ptrace_nelt != 0)fatal("bad pipetrace args, use: ");
    fu_pool = res_create_pool("fu-pool", fu_config, N_ELT(fu_config));
    rslink_init(MAX_RS_LINKS);
    tracer_init();
    fetch_init();
    cv_init();
    eventq_init();
    readyq_init();
    ruu_init();
    lsq_init();
    dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

○ A
○ B
○ No preference

Figure C-12   Case 12   Group - Snippet 3



A

```
int i, j, index, n_std_unknowns;
md_addr_t std_unknowns[MAX_STD_UNKNOWNS];
for (i=0, index=LSQ_head, n_std_unknowns=0;i < LSQ_num; i++, index=(index + 1) % LSQ_size)
{
        if ( (MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE))
        {
                if (!STORE_ADDR_READY(&LSQ[index]))
                {
                        break;
                }
                else if (!OPERANDS_READY(&LSQ[index]))
                {
                        if (n_std_unknowns == MAX_STD_UNKNOWNS)
                                fatal("STD unknown array overflow, increase MAX_STD_UNKNOWN
                        std_unknowns[n_std_unknowns++] = LSQ[index].addr;
                }
                else
                {
                        for (j=0; j < n_std_unknowns; j++)
                        {
                                if (std_unknowns[j] == LSQ[index].addr)std_unknowns[j] = 0;
                                if (rs->in_LSQ && ((MD_OP_FLAGS(rs->op) &
                                      (F_MEM|F_STORE)) == (F_MEM|F_STORE)))
                                {
                                      rs->issued = TRUE;
                                      rs->completed = TRUE;
                                      if (rs->onames[0] || rs->onames[1])
                                      {
                                            panic("mis-predicted store");
                                            ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, 
                                            n_issued++;
                                      }
                                }
                        }
                }
        }
        if ( ((MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_LOAD)) == (F_MEM|F_LOAD))&& !LS
                && !LSQ[index].issued && !LSQ[index].completed && OPERANDS_READY(&LSQ[
        {
                for (j=0; j            {
                        if (std_unknowns[j] == LSQ[index].addr) break;
                }
                if (j == n_std_unknowns)
                {
                        readyq_enqueue(&LSQ[index]);
                }
        }
}
```

B

```
int i, j, index, n_std_unknowns;
md_addr_t std_unknowns[MAX_STD_UNKNOWNS];
for (i=0, index=LSQ_head, n_std_unknowns=0;i < LSQ_num; i++, index=(index + 1) % LSQ_size)
{
  if ( (MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE))
  {
    if (!STORE_ADDR_READY(&LSQ[index]))
    {
      break;
    }
    else if (!OPERANDS_READY(&LSQ[index]))
    {
      if (n_std_unknowns == MAX_STD_UNKNOWNS)
        fatal("STD unknown array overflow, increase MAX_STD_UNKNOWNS");
      std_unknowns[n_std_unknowns++] = LSQ[index].addr;
    }
    else
    {
      for (j=0; j < n_std_unknowns; j++)
      {
        if (std_unknowns[j] == LSQ[index].addr)std_unknowns[j] = 0;
        if (rs->in_LSQ && ((MD_OP_FLAGS(rs->op) & (F_MEM|F_STORE)) == (F_MEM|F_S
        {
          rs->issued = TRUE;
          rs->completed = TRUE;
          if (rs->onames[0] || rs->onames[1])
          {
            panic("mis-predicted store");
            ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK, 0);
            n_issued++;
          }
        }
      }
    }
  }
  if ( ((MD_OP_FLAGS(LSQ[index].op) & (F_MEM|F_LOAD)) == (F_MEM|F_LOAD))&& !LSQ[ind
    && !LSQ[index].issued && !LSQ[index].completed && OPERANDS_READY(&LSQ[index]))
  {
    for (j=0; j      {
      if (std_unknowns[j] == LSQ[index].addr) break;
    }
    if (j == n_std_unknowns)
    {
      readyq_enqueue(&LSQ[index]);
    }
  }
}
```

Which version of the formatted code snippet do you find the most aesthetically pleasing?

- A
- B
- No preference

Submit Selection

Withdraw from Survey

Figure C-13 Case 13 Indent - Snippet 4