AUTOMATED SOURCE CODE STRUCTURE FEEDBACK USING SRCML AND RELAXNG

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Brandon M. Sedgwick

August, 2013
ABSTRACT

Students learning programming in a Computer Science course, and software developers in industry both benefit from immediate feedback. Dynamic validation, such as software testing, tests the behavior of code and is often full automated to provide this kind of feedback. Static validation checks the structure of the source code, however very few automated tools exist that can practically perform this validation. The research presented is a framework for validating the structure of source code. The source code is converted into srcML, an XML format for source code, and the source code structure is validated using the XML grammar language RelaxNG. An evaluation was performed where the automated approach was applied to the source code written in twelve labs for a first course in programming of over 300 students. The implementation described here was found to be highly scalable and flexible. In academics, applications of the work include automated grading and student feedback, and in industry to code reviews, code policy enforcement, and code pattern learning.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
</tr>
<tr>
<td>II. COMPUTER SCIENCE LABS</td>
</tr>
<tr>
<td>III. RELAXING AND AUTOMATION</td>
</tr>
<tr>
<td>IV. EXPERIMENTATION AND RESULTS</td>
</tr>
<tr>
<td>V. FUTURE WORK</td>
</tr>
<tr>
<td>VI. CONCLUSION</td>
</tr>
</tbody>
</table>

| BIBLIOGRAPHY | 72 |

| APPENDIX | 74 |
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Lab 1 Source Code: Account Setup and the Programming Environment</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Lab 2 Source Code: Parts of a Program</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Lab 3 Source Code: Output Formatting and Software Library Headers</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Lab 4 Source Code: Conditional Branching</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Switch Statement srcML</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>Lab 5 Source Code: More Conditional Branching</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Lab 6 Source Code: Loops</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>Lab 7 Source Code: Function</td>
<td>19</td>
</tr>
<tr>
<td>2.9</td>
<td>Lab 8 Source Code: More Functions</td>
<td>21</td>
</tr>
<tr>
<td>2.10</td>
<td>Lab 9 Source Code: Arrays</td>
<td>24</td>
</tr>
<tr>
<td>2.11</td>
<td>Lab 10 Source Code: More Arrays</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>Lab 11 Source Code: Pointers</td>
<td>29</td>
</tr>
<tr>
<td>2.13</td>
<td>Lab 12 Source Code: Structs</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Simple Logic Concepts of RelaxNG</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Attributes, Values, and Empty RelaxNG Elements</td>
<td>35</td>
</tr>
</tbody>
</table>
3.3 Usings Definition RelaxNG
3.4 Comments Definition RelaxNG
3.5 Include Statement in srcML
3.6 Includes Definition RelaxNG
3.7 Return Definition RelaxNG
3.8 Expression Definition RelaxNG
3.9 Declarative Statement RelaxNG
3.10 Declarative Definition RelaxNG
3.11 If Definition RelaxNG
3.12 If Block Definition RelaxNG
3.13 Switch Definition RelaxNG
3.14 While Loop Definition RelaxNG
3.15 For Loop Definition RelaxNG
3.16 Do While Loop srcML
3.17 Do While Loop Definition RelaxNG
3.18 Function Call Definition RelaxNG
3.19 Function Declaration Definition RelaxNG
3.20 Function Definition RelaxNG
3.21 Core RelaxNG Include RelaxNG File
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Complete Automation Process</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>Lab 1 srcML</td>
<td>62</td>
</tr>
<tr>
<td>4.3</td>
<td>Sample Source Grammar Output</td>
<td>66</td>
</tr>
<tr>
<td>4.4</td>
<td>Source Grammar Validation Run Times</td>
<td>67</td>
</tr>
<tr>
<td>A.1</td>
<td>Lab 1 RelaxNG</td>
<td>74</td>
</tr>
<tr>
<td>A.2</td>
<td>Lab 2 srcML</td>
<td>76</td>
</tr>
<tr>
<td>A.3</td>
<td>Lab 2 RelaxNG</td>
<td>78</td>
</tr>
<tr>
<td>A.4</td>
<td>Lab 3 srcML</td>
<td>80</td>
</tr>
<tr>
<td>A.5</td>
<td>Lab 3 RelaxNG</td>
<td>84</td>
</tr>
<tr>
<td>A.6</td>
<td>Lab 4 srcML</td>
<td>86</td>
</tr>
<tr>
<td>A.7</td>
<td>Lab 4 RelaxNG</td>
<td>89</td>
</tr>
<tr>
<td>A.8</td>
<td>Lab 5 srcML</td>
<td>91</td>
</tr>
<tr>
<td>A.9</td>
<td>Lab 5 RelaxNG</td>
<td>96</td>
</tr>
<tr>
<td>A.10</td>
<td>Lab 6 srcML</td>
<td>98</td>
</tr>
<tr>
<td>A.11</td>
<td>Lab 6 RelaxNG</td>
<td>102</td>
</tr>
<tr>
<td>A.12</td>
<td>Lab 7 srcML</td>
<td>105</td>
</tr>
<tr>
<td>A.13</td>
<td>Lab 7 RelaxNG</td>
<td>108</td>
</tr>
<tr>
<td>A.14</td>
<td>Lab 8 srcML</td>
<td>111</td>
</tr>
<tr>
<td>A.15</td>
<td>Lab 8 RelaxNG</td>
<td>114</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Currently being taught in software engineering and computer science courses is the combination of program output through data structures and algorithms with the focus of source code and version control. Program output has been given the simple task to automate the validation and verification. This automation has been modified to provide numerous methods for outputting results. For education and industry development, results from testing output are used daily to show the current status of the programs and applications. However, current feedback for source code and applications is limited to human reliability for manually grading and checking of source code structure to provide feedback to students. In addition to this limitation, tools and processes to aid in control and agility would be version control. The combination of version control and automated tools can give education a tremendous boost in productivity.

The need for source code validation stretches from software engineering education to industry software development processes. Version control usage, although a separate process, complements automated processes in software engineering education. Currently, and traditionally, programs have been submitted once the assignment has been completed and manually graded for how the code was written. Testing output of programs has been extensibility covered by many testing techniques. However, the internal structure of how
code is written is far less covered by automation. A programmatic tool would be needed to solve this current problem.

A validation tool has been developed to translate raw source code, like C++ or Java, into srcML representations and validate the XML against a corresponding RelaxNG grammar file. The results from running this tool will offer insight into what future research can be derived from this process. Using source code from several hundred students in introductory computer science classes, the validation done was simply focused on the expected internal code structure of their programs. Most importantly, the results will show that the traditional manual grading of source code could be accurately automated using appropriate tools, techniques, and processes. This paper will discuss the research applied to automating the validation of source code structure and providing feedback for it using this tool.

Using the srcML toolkit, source code can easily be translated into nicely formatted XML. As shown by Collard et. al., the srcML toolkit can be utilized for fast, lightweight transformations and analyses [3]. From this resulting srcML as XML, standard tools to translate and validate the XML can be applied with limitless possibilities. Tools used for translating XML documents include XSLT, XPath, XQuery, and many more. Validating XML documents with grammars like tools using RelaxNG, DTD, XSD, and SOX focus on the absence or the presence of particular nodes on the XML documents.

RelaxNG as a grammar offers a simpler approach to XML validation as compared to techniques with XSD, SOX, and DTD. Developed with an XML-based schema language in parallel with a compact syntax schema, RelaxNG provides an easier to read grammar language for creating schema files to validate any XML formatted document. The
grammar can be composed of a single file or a set of multiple files. Like XML, RelaxNG is structured in elements. Groups of elements can be abstracted out to definitions, which can be referenced by name like a variable. As with validation of XML with other techniques and tools, RelaxNG can check for the presence and absence of elements.

Making this research possible, the University of Akron's Computer Science department supplemented introductory classes with version control starting in the Fall of 2011. The weekly process of labs to students started in an automated fashion of distributing the source code to all of the students' code bases in the version control repository. Instructions were provided through wiki pages. This process started by enforcing iterative development to complete labs each week. Labs were broken down to distinct steps, typically consisting of 4 to 5 steps per lab. For completion of a step, a commit message could consist of a description or tag like “Step 1: ...” with a message about what changes were made.

Once the due date had passed, reports were automatically created. These reports show details from testing results, the actual source code, subversion commit message history for the lab, and meta data like name, class, section, date, etc. For testing results, a separate test file is created with the testing output. That file's data is read to create the lab report.
CHAPTER II

COMPUTER SCIENCE LABS

The experimental data used for the research presented in this paper came from the labs developed for the University of Akron Computer Science Department’s introductory course, Computer Science 1, from the fall of 2011 to the fall of 2013. These courses consisted of 12 labs to introduce students to C++ code and complete applications. Work was also completed for the Computer Science 2 labs as well, but those labs were out of scope for the work of this research. Both sets of labs were developed, instructed, graded, and maintained by the author. These sets of labs include Computer Science 1 in the semesters of Fall 2011\textsuperscript{1}, Spring 2012\textsuperscript{2}, and Fall 2012\textsuperscript{3} as well as Computer Science 2 in the semesters of spring 2012 and fall 2012. One caveat during the development of this set of introductory labs was that the curriculum was modified for Fall 2012 to exclude structs and classes, which were moved to Computer Science 2.

\textsuperscript{1} https://dev.cs.uakron.edu/trac/CS1F2011
\textsuperscript{2} https://dev.cs.uakron.edu/trac/cs209s12
\textsuperscript{3} https://dev.cs.uakron.edu/trac/cs209fa12
2.1 Lab 1: An Introduction to Coding and Version Control

The introductory course to computer science starts out with the first lab that focuses on simple IDE setup, version control setup, and composing a Hello World C++ application. Above all, the introductory lab emphasizes the usage of version control, which makes sharing of source code possible. Subversion is a tool for saving source code or any type of file to a centralized repository, or server, that allows other people access to it depending on the authentication. Subversion allows for simple editing of a text file.

For a student, the commands and workflows needed for contributing to their source code repositories was relatively simple compared to a typical developer working on a team. To initially get their source code repository as a local copy, each student was instructed to execute the `checkout` command, which clones what is in the central repository to their local copy. Then as they make changes to their local copy of source code, the `commit` command will push up any outstanding changes to the tracked files. To illustrate this concept better, the students were instructed and informed how the web browser for the URL of the central repository will show what the current version has as plain text for each file.

The source code provided to the students was an introductory program that is used in nearly every language one will learn, the Hello World program. As seen in Figure 2.1, the program only makes a calculation writing hard coded values and outputs the result to standard output. When processing the source code when finished, the script processed the source code using the srcML toolkit for the XML representation. Applying the validation tool I created against the RelaxNG grammar for Lab 1 validated whether or not
the student had the necessary elements in their source code. In a sense, the Lab 1 RelaxNG file was a “Hello World” program with only the basic elements of a program.

```cpp
/*******************************
* Lab 1 : HelloLab.cpp
* Author: Some Student
* student@zips.uakron.edu
* Purpose: Demonstration of a simple program
*******************************
#include <iostream>
using namespace std;

int main() {
    // variables for payrate and hours
    int payrate = 7;
    int hours = 40;
    int totalpay = payrate * hours;
    // Prints welcome message...
    cout << "Total pay: $" << totalpay << ".00 " << endl;
    return 0;
}
```

Figure 2.1 - Lab 1 Source Code: Account Setup and the Programming Environment

Validating the source code translated into XML from the srcML toolkit against the RelaxNG grammars, there were only a couple of elements that comprise the complete document. Before the main function, a source code file would be considered valid if it contained a header comment, include iostream statement, using statement, and any other white space or comments. Then within the main function, a combination of comments and expression statements is followed by a return statement. When an element was present that was not expected, the output showed the line and reason for the violation. As seen in the excerpt of RelaxNG in XML syntax below, the main function block is only composed of just a couple of XML elements.
The "ref" elements are references to external grammar files with those definitions abstracted out for optimal reuse.

2.2 Lab 2: Demonstrating Parts of a Program

Following the introductory lab that got students familiar with all the tools being used, the second lab outlined and explained the components of a simple C++ program. Starting with an empty source code file, the students had to edit the source code file, compile the program, run the program, check the output, and make a commit to the subversion repository at each step. Aside from teaching the students the basics of coding using C++, the process of editing the source code, testing its output, and committing incrementally was the emphasis to be instilled for development. As seen in Figure 2.2, the source code for Lab 2 shows the various expression statements to make calculations and display output to the console.
/*
   DriveTime.cpp

   Calculate the time it will take to drive given a
distance in miles and a speed in miles per hour (mph)

   Some Student
   student@zips.uakron.edu
*/

#include <iostream>

using namespace std;

int main() {

   // variables to hold distance and speed
   double distance1 = 100;
   double distance2 = 50;
   double speed = 35;

   // input the distance in miles
   cout << "Enter two integer distances (in miles) :" << endl;
   cin >> distance1 >> distance2;

   // input a speed in mph
   cout << "Enter an integer for the travel speed (in mph) :" << endl;
   cin >> speed;

   // variable to hold resulting time
   double traveltime = 3;

   // calculate the drive time based on distance and speed
   traveltime = (distance1 + distance2) / speed;

   // output the drive time
   cout << " The travel time is " << traveltime << " hours. " << endl;

   return 0;
}

Figure 2.2 - Lab 2 Source Code: Parts of a Program
2.3 Lab 3: Program Output and Formatting

Lab 3 continued with what was learned in the first two labs, while adding the concepts of including libraries, calling library functions, and using limited objects like cin and cout. From a source code perspective, the students were asked to add include statements for libraries like iostream, iomanip, cmath, and string. As seen below, the srcML to include a library uses the cpp namespace for its elements like include, directive, and file.

```xml
<cpp:include>#<cpp:directive>include</cpp:directive><cpp:file>&lt;string&gt;</cpp:file></cpp:include>
```

Another vital expression added was the function call. As seen in Figure 2.3, function calls were used like getline(cin, person1) and abs(score1 – average). To demonstrate the added complexity for function calls in the resulting XML, the following is the srcML for the getline function call referred to.

```xml
<expr_stmt><expr><call><name>getline</name><argument_list>(
  <argument><expr><name>cin</name></expr></argument>,
  <argument><expr><name>person1</name></expr></argument>)</argument_list>
</call></expr>;</expr_stmt>
```

As seen in this XML, many arguments could be present in a function call expression.

With numerous variations for possible function calls that may have appeared in students’ resulting source code, many patterns had to be considered for the RelaxNG grammar. One design decision made was to separate out the expression into its own pattern. Therefore, the expression statement, expr_stmt tag in srcML and argument could utilize this expression grammar pattern.
Figure 2.3 - Lab 3 Source Code: Output Formatting and Software Library Headers
2.4 Lab 4: Conditional Branching

At this point in the series of labs, a new structure to programs was introduced with if statements. First, the students were taught from the ground up by explaining mathematical comparison operators that produce a boolean value of true or false. From there, the boolean operators were put into context within if statements. Since the structure of if statements requires blocks, the internal structure abstracted out is a similar pattern to a function definition, which includes the function block in addition to its supplementary elements like function name, parameter list, return type, and other attributes. So far, the grammars created all included one function defined – the main function. The else part of the if-else statement would also be very similar as well by having the following node in the structure be the else element with its own block child element.

The if statement as a code structure is comprised of the if keyword, condition or test, and block. Within the block of the if statement, any number of statements, expressions, and white spaces may be present. As seen in Figure 2.4, the students went as far as the if-else statements and nested if statements. In the source code structure for nested if statements and if-else, the else or else if keywords would follow the block of the previous if or else if and have their own block.
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main() {
  // Constant symbols to use throughout the program.
  const int SUN = 1, MON = 2, TUE = 3, WED = 4, THU = 5, FRI = 6, SAT = 7;
  const int JAN = 1, FEB = 2, MAR = 3, APR = 4, MAY = 5, JUN = 6, JUL = 7, AUG = 8, SEP = 9, OCT = 10, NOV = 11, DEC = 12;

  // variables to hold input.
  int month, day;
  string person;

  // prompt user to enter a name
  cout << "What is your name?\n"; 
  getline(cin, person);

  // prompt user to enter month
  cout << "Type a number " << JAN << "-" << DEC << " to select a month:\n";
  cin >> month;
  cin.ignore(20000, '\n');

  // prompt user to enter day of the week
  cout << "Type a number " << SUN << "-" << SAT << " to select a day (Sunday - Saturday)\n";
  cin >> day;
  cin.ignore(20000, '\n');

  cout << "You chose month: " << month << endl;
  cout << "You chose day: " << day << endl;

  // Part 1: compound conditional
  cout << "------ part 1: ------" << endl;   // leave this line here
  if ((day = 7) || (day = 1))  {
    cout << "It is a weekend!" << endl;
  }

  // Part 2 and 3: if/else
  cout << "------ part 2 and 3: ------" << endl;   // leave this line here
  if  ((month >= 5 ) && (month <= 8 )) {
    cout << "Summer semester" << endl;
  } else if  ((month >=1) && (month <= 4)) {
    cout << "Spring semester" << endl;
  } else if  ((month >= 9) && (month <= 12)) {
    cout << "Fall semester" << endl;
  } else {
    cout << "INVALID MONTH" << endl;
  }

  // Part 4: nested 'if'
  cout << "------ part 4: ------" << endl;   // leave this line here
  if ((month >= 5 ) && (month <= 8 )) { 
    cout << "A summer semester" << endl;
  } else if ((month >= 13) || (month < 1)) {
    cout << "an invalid month" << endl;
  } else if ((month <= 4) || (month >= 8) ) {
    cout << "A non-summer semester" << endl;
    if ((day = 7) || (day = 1)){
      cout << "It is the weekend!" << endl;
    } else if ((day = 2) && (day <= 6)) {
      cout << "It is a weekday." << endl;
    } 
  }

  return 0;
}
2.5 Lab 5: More Conditional Branching

Building off of what Lab 4 instructed students to implement with conditional branching using if else statements, the instructions for Lab 5 include switch statements, ternary operators, and more nesting of if statements. Of the two structures introduced in this lab, the switch statement and ternary operator in an expression, the switch statement was the only significant change in srcML output although the ternary operator expressions were deemed just as difficult by the students.

The switch statement code structure was a different form for a logical control structure in the sequence of statements learned. As seen here, the srcML for the switch statement in Figure 2.5 is set up similar to an if statement with the condition and block following it. Then within the switch’s block element, there are many case statements followed by the one default, which each could have expressions, statements, and comments.
As seen above in Figure 2.5, the switch statement indicates that each case statement contained an expression. Like the abstraction from creating the RelaxNG patterns for Lab 3, the case statements, expression statements, arguments, and conditions of if statements used the expression pattern to this point in the series of labs.
Figure 2.6 – Lab 5 Source Code: More Conditional Branching
2.6 Lab 6: Loops

The various loops have all the similar code structures. Between while, for, and do while loops, each has a condition as a control to the loop and a block for the logic to be done. As described to the students familiar with if statements, the while loop has the same structure as the if statement but with a different key word. The block within both the if statement and while loop executes, but the while loop during run time iterates over its execution block as the condition passes. Very similar to the while loop, the do while loop is the same structure but flipped around with the condition following the block. The for loop adds additional complexity to the control to the loop with three parts – initialization, condition, and update.

In Figure 2.7, a code snippet is shown from Lab 6 to demonstrate a couple loops that the students were asked to create.
# Lab 6: Loops.cpp

* Author: Some Student
  * student@zips.uakron.edu

* Purpose: Demonstrate conditional loop structures

#include <iostream>
using namespace std;

// This is a function prototype.
// We will discuss them in later labs, but ignore it for now.
int input_an_int();

int main() {
  // multipurpose reusable integer for loop counting.
  // remember to initialize before each use!
  int i;

  // multipurpose reusable integer to hold user input.
  int num;

  // Part 1: off-by-one
  cout << "------ part 1 (off by one): ------" << endl;
  num = input_an_int(); // function call
  i = 0;
  // this will loop from 0 to (n-1)
  while(i <= num) {
    cout << i << " 
";
    ++i;
  }
  cout << endl;

  // Part 2: accumulate a sum
  int newnum;
  cout << "Please enter another number for second loop.
" ; //user prompt
  i = num;
  int numtotal = 0;
  while ( i > 0 ) {
    if ( i <= num ) {
      cout << i << " 
";
      numtotal = i + numtotal;
    } else {
      cout << endl;
    }
  }
  cout << "the sum of 0 to " << num << " is " << numtotal;

  // Part 3: for loop
  int num = input_an_int();
  for ( i= 0 ; i * i <= num ; ++i ) {
    cout << (i * i) << " 
";
  }

  // Part 4: do-while
  // Part 5: nested loops
}

Figure 2.7 – Lab 6 Source Code: Loops
2.7 Lab 7: Functions

Using functions allows abstractions in software development, which also promotes reuse. As in developing for mathematics, encapsulating a large formula into a single function call that can be treated like an atomic operation. The first concepts that were taught to the students include defining functions and using those functions. In the case of the introductory lab on functions, there was a single file without class files and header files.

Being asked to only implement basic functions, the code expected was very predictable. The parameters and return types were predefined in the instructions for the students to implement. Although some order of statements and comments can vary, most of the possible code elements are limited to expression statements, declarative statements, comments, and whitespace.
# Lab 7: Functions.cpp

* Author: Some Student
  * student@zips.uakron.edu

* Purpose: Introduce concepts of functions

*********************************
#include <iostream>
#include <string>
using namespace std;

// prototypes
int input_an_int (); //part one prototype
void print_sequence (const int);//part two prototype
string rate_time (const int); //part three of doom returning a value
int call_counter (); // part five static local variables

int main() {
  int n; // for user input
  // the variable that will count
  cout << "Sequence from 0 to end\n";
  n = input_an_int();
  call_counter (); //part 5
  print_sequence (n);
  call_counter (); //part 5
  /****part 3******/
  cout << "Please enter your commute time in minutes\n";
  cin >> time;
  rate_time (time);
  call_counter (); //part 5
  return 0;
}

// Get user input and try to interpret it as an integer.
int input_an_int () {
  cout << "Type an integer, then press enter:"
  >> flush;
  int num = 0;
  cin >> num;
  cin.ignore(20000, '\n');
  return num;
}

void print_sequence (const int n) {
  cout << "Sequence from 0 to " << n << ":
  for(int i=0;i<=n;++i){
    cout << i << " ";
  }
  cout << endl;
}

string rate_time (const int time) {
  if (time > 0 && time <= 30) {
    return " is a weak commute!";
  } else if (time > 30 && time <= 60) {
    return " minutes a long - short = medium commute.";
  } else if (time > 60 ) {
    return " is a long commute! Play the dub step!";
  }
}

int call_counter (){
  static int x = 0;
  x++;
  cout << "I have been called " << x << " times\n";
  return x;
}
2.8 Lab 8: More Functions

In addition to the brief overview of functions in the previous lab, the functions in Lab 8 concentrated on parameter types and parameter passing. For the students, the lab went through basic functions, functions with default parameters, function overloading, and passing arguments to a function by reference. For the first new structure element introduced for functions, default parameters set an initial value to a parameter being passed into the function, as seen here:

```c
void some_function(int a, int b = 0)
```

Function overloading did not contribute to new code structural elements but only a new concept of using the same function name with various sized and typed parameter lists. The last part of this lab introduced another functional element, parameters passed by reference. As seen in this code snippet below, all three parameters are passed by reference:

```c
void input_int(int & m, int & n, int & o)
```
# File: Parameters.cpp
# Author: Some Student
# some@zips.uakron.edu
# Purpose: Demonstrate advanced function parameter concepts
#include <iostream>
using namespace std;

// prototypes
int input_int();
int input_int();
int print_sequence (int, int, int);
int print_sequence (double, double, double);

int main() {
    // for user input
    int m, n;
double dm, dn;
    int step = 1;

    /*
     * Output sequence from startval to endval, entered by user
     */
    cout << "Integer sequence from BEGIN to END and STEP value.\n";
    m = input_int();
n = input_int();
step = input_int();
print_sequence (m, n, step);
}

// Get user input and try to interpret it as an integer.
int input_int() {
    int num = 0;
    cout << "Type an integer, then press enter:\n";
    cin >> num;
    cin.ignore(20000, '\n');
    return num;
}

int print_sequence (int m, int n, int step ) {
    cout << "Int sequence from " << m << " to " << n << ": ";
    for(int i = m; i <= n; i += step){
        cout << i << " ";
    }
    cout << endl;
    return 0;
}

double print_sequence (double dm, double dn, int step ) {
    // move the following lines into a new function:
    cout << "Int sequence from " << dm << " to " << dn << "\n";
for(double i = dm; I <= dn; i += step) {
    cout << i << " ";
}
    cout << endl;
    return 0;
}
2.9 Lab 9: Arrays

For Lab 9, the students were presented a base for understanding data structures with the introduction of arrays. As described in the lab instructions, an array is a way to use a single name to refer to a set of values of the same datatype.

“Back in Lab3, you created four variables to hold test scores for four imaginary students. Normally, a class has more than four students. We can use an array to hold these scores instead of declaring 40 separate named variables. The values in an array are laid out contiguously in memory (they are all adjacent to each other, and in order).”

To complete the lab, steps were created to initialize an array, access an array as a zero-based ordering with off-by-one concerns, passing an array to a function, modifying an element within the array, and iterating through an array. The source code showing these various steps is shown in Figure 2.10.

Derived from the source code, the srcML representation had mostly the same source code structure aside from the subtle differences for arrays. Below is the srcML representation of the line below “Do Part 1…” in Figure 2.9.

---

4 [https://dev.cs.uakron.edu/trac/cs209s12/wiki/Lab9](https://dev.cs.uakron.edu/trac/cs209s12/wiki/Lab9)
As seen here, the declarative statement has an *index* element following the *type* and *name*.

For the initialization block, the initialized values are simply a list of expressions.
#include <iostream>
using namespace std;

void print_array (double [], int);

int main() {
    const int ARRAYLEN = 10;

    /* Do part 1 here: */
    double numbers[ARRAYLEN] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};

    /* Do part 2 here: */
    cout << numbers [9] << endl;

    /* Do parts 3 and 4 here: */
    print_array (numbers, ARRAYLEN);

    /* Do part 5 here: */
    double mult = 10.0;
    for (int i2 = 0; i2 < ARRAYLEN ; ++i2 ) {
        numbers [i2] = numbers[i2] * mult;
    }
    print_array (numbers, ARRAYLEN);

    /* Do part 6 here: */
    double value1;
    for (int i3; i3 < ARRAYLEN; ++i3) {
        value1 = numbers[i3] + value1;
    }
    cout << "Sum of array elements: " << value1 << endl;

    /* Do part 7 here: */
    char who[] = "Charles Babbage";
    cout << who << endl;
    return 0;
}

void print_array (double numbers[], const int ARRAYLEN) {
    for(int i = 0; i < ARRAYLEN; ++i ) {
        cout << numbers[i] << " ";
    }
    cout << endl;
}
2.10 Lab 10: More Arrays

Similar to Lab 9, Lab 10 continued reinforcing operations with arrays. Most of the first sections discussed iterating over a single dimensional array, which was the same principle and code structure as before. However, the difference in code structure came in the final part of the lab with two dimensional arrays.

For the following line of code, a two dimensional array is initialized with values.

```java
int matrix[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12}
```

Then, using the srcML toolkit that line of code is translated to the following XML.

```xml
<decl_stmt>
  <decl><type><name>int</name></type> <name><name>matrix</name>
  <index>
    <expr>3</expr> </index>
  <index>
    <expr>4</expr> </index>
  </name>=<init> <expr><block>{<expr>1</expr>,<expr>2</expr>,<expr>3</expr>,<expr>4</expr>,
    <expr>5</expr>,<expr>6</expr>,<expr>7</expr>,<expr>8</expr>,<expr>9</expr>,<expr>10</expr>,
    <expr>11</expr>,<expr>12</expr>}</block></expr></init></decl>
</decl_stmt>
```

As in the Lab 9 srcML result of the single dimensional array, the two dimensional array just adds a second index tag with an expression within it.
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

/* implement for part 0 */
void print_array(int arr[], int len) {
    for (int i = 0 ; i < 5 ; i++) {
        cout << arr[i] << " ";
    }
}

/* implement for part 1 */
void input_array(int inputs[], int ARRAYLEN) {
    for (int pie = 0 ; pie < ARRAYLEN ; pie++)
        cin >> inputs[pie];
}

/* implement for part 2 */
bool arrays_equal(int inputs[], int numbers[], int ARRAYLEN) {
    for (int count = 0 ; count < ARRAYLEN ; count++ ) {
        if (inputs[count] != numbers[count]) {
            return 0;
        }
    }
    return 1;
}

/* implement for part 3 */
void print_table(string people[], int inputs[], int ARRAYLEN) {
    for (int o = 0 ; o < ARRAYLEN ; o++) {
        cout << people[o] << right << setw (11);
        cout << inputs[o] << left << setw (11) << endl;
    }
}

/* implement for part 4 */
void print_array(int arr[][4], int h, int w) {
    for (int i2 = 0 ; i2 < h ; i2++ ) {
        for(int i5 = 0 ; i5 < w ; i5++ ) {
            cout << arr[i2][i5] << " ";
        }
        cout << endl;
    }
}

int main() {
    const int ARRAYLEN = 5;
    int numbers[ARRAYLEN] = {1,4,9,16,25};
    int inputs[ARRAYLEN];
    string people[] = {"Big Bird","Bert","Ernie","Grover","Oscar"};

    print_array(numbers, ARRAYLEN);
    cout << "---- Part 1: passing arrays ----\n";
    cout << "Enter " << ARRAYLEN << " integers:";
    input_array(inputs, ARRAYLEN);
    print_array(inputs, ARRAYLEN);
    cout << "---- Part 2: comparison ----\n";
    if(arrays_equal(numbers, inputs, ARRAYLEN)) {
        cout << "Arrays are equal" << endl;
    } else {
        cout << "Arrays are not equal" << endl;
    }
    cout << "---- Part 3: parallel arrays ----\n";
    print_table(people, inputs, ARRAYLEN);
    cout << "---- Part 4: 2D arrays ----\n";
    int matrix[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
    print_array(matrix, 3, 4);
    return 0;
}
2.11 Lab 11: Pointers

To this point in time, all labs were straight-forward code with no direct memory manipulation, but Lab 11 introduced students, by the title of the lab, pointers. Pointers in C++ allowed students many more opportunities to create data structures and control of their variables. The following background information was provided to the students in the lab.

“Every variable in your program exists in system memory. System memory is indexed by address (memory location). So, each variable has an address. A pointer is a type of variable meant to hold a memory address. (since it is a variable, it also exists at its own address, but we will not go that deep) Every pointer must also specify whatever datatype it can point to: You can have a "pointer to integer" or "pointer to double", etc.”

The code structure for pointers is very similar to simple variable declarations and variable initializations. The following declaration in C++ showed students how easy it was to create a pointer.

```c
int *ptr;
```

From that declaration using the srcML toolkit, the resulting XML is as follows.

```xml
<decl_stmt>
  <decl>
    <type>
      <name>int</name>*</type>
    <name>ptr</name>
  </decl>
</decl_stmt>
```

The srcML for the pointer's declaration is the same code structure, but can be recognized by the pointer notation, *, within the declaration's type but after the type's name. Furthermore, the initialization of the pointer variable described before is shown below.

```c
ptr = &some_integer;
```

---

5 https://dev.cs.uakron.edu/trac/cs209s12/wiki/Lab11
Similarly, the variable initialization looks like any other except for the `&` found before the name of the variable on the right.

```
<expr_stmt><expr><name>ptr</name> = &<name>some_integer</name>
</expr>;</expr_stmt>
```
```cpp
#include <iostream>
using namespace std;

// function prototypes
void print_array(int *, int);
void print_address_and_value(const int *);

int main() {
    int x = 2; int *ptr; // declaring pointer to int
    const int SEQUENCE_LENGTH = 8; int seq[SEQUENCE_LENGTH] = {10,20,30,40,50,60,70,80};
    // Part 1: get an address
    cout << "---- Part 1: get an address ----
";
    ptr = &x;
    cout << "ptr now holds the address: " << ptr << endl;
    cout << endl;
    // Part 2: set a dereferenced value
    cout << "---- Part 2: dereferencing ----
";
    cout << *ptr << endl;
    cout << "x is: " << x << endl;
    cout << endl;
    // Part 3: alias an array
    cout << "---- Part 3: alias an array ----
";
    ptr = seq;
    cout << "value of seq (location of array data): " << seq << endl;
    cout << "value of ptr (where ptr points): " << ptr << endl;
    print_array(ptr, SEQUENCE_LENGTH);
    cout << endl;
    // Part 4: pointer arithmetic
    cout << "---- Part 4: pointer arithmetic ----
";
    for (int a = 0; a < SEQUENCE_LENGTH; a++) {
        int *looptr = seq + a;
        print_address_and_value(looptr);
    }
    cout << endl;
    // Part 5: const ptr vs. ptr to const
    cout << "---- Part 5: pointers and constants ----
";
    x = 10;
    int * const annoying_ptr = &x;
    cout << "x (before) is: " << x << endl;
    print_address_and_value(annoying_ptr);
    *annoying_ptr = 20;
    cout << "x (after) is: " << x << endl;
    print_address_and_value(annoying_ptr);
    cout << endl;
    // Part 6: dynamic allocation
    cout << "---- Part 6: dynamic allocation ----
Input a desired array size
";
    cin >> x;
    ptr = new int[x];
    for(int i=0; i<x; ++i) {
        ptr[i] = i;
    }
    print_array(ptr, x);
    delete [] ptr;
    return 0;
}

// utility functions. do not modify.
void print_address_and_value(const int *p) {
    cout << p << " : " << *p << endl;
}
void print_array(int ar[], int len) {
    for(int i=0; i<len; ++i) {
        cout << ar[i] << " ";
    }
    cout << endl;
}
```
2.12 Lab 12: Structs

The last of the introductory labs presented to students before classes was the Lab 12 on structs. When the source for a struct was translated by the srcML toolkit, the struct introduced a new XML tag, `struct`. As with the various other elements like functions, loops, if-else, and switch statements, the structs have a block element to them that contains the variable declarations. For the following code sample from Figure 2.12, a simple struct is defined with a couple elements within it.

```c
struct Student
{
    string name;
    Place pData;
    int student_id;
    double total_mark;
};
```

Once processed by the srcML toolkit and translated to XML, the following is the result.

```xml
<struct>struct <name>Student</name>
    <block>{<public type="default">
        <decl_stmt><decl><type><name>string</name></type>
            <name>name</name></decl>;</decl_stmt>
        <decl_stmt><decl><type><name>Place</name></type>
            <name>pData</name></decl>;</decl_stmt>
        <decl_stmt><decl><type><name>int</name></type>
            <name>student_id</name></decl>;</decl_stmt>
        <decl_stmt><decl><type><name>double</name></type>
            <name>total_mark</name></decl>;</decl_stmt>
    </public>}</block>
</struct>
```

As seen above, the `struct` element contains the struct keyword, the struct name, and the struct's block with the variable declarations.
# Lab 12: Structs.cpp

* Author: Some Student  
  student@zips.uakron.edu  
* Purpose: Demonstrate the use of structs

```cpp
/********************************
* Lab 12: Structs.cpp
*
* Author: Some Student
*         student@zips.uakron.edu
* Purpose: Demonstrate the use of structs
********************************/
#include <iostream>
using namespace std;

struct Place
{
    string student_address;
    string student_city;
    string student_state;
};

/** Part 1: Declaration of structure ****/
struct Student
{
    string name;
    Place pData;
    int student_id;
    double total_mark;
};

//function prototypes
void print_structure(Student);
void print_structure_using_pointer(Student);
void print_nested_structure(Student);

int main()
{
    /**** Part 2: Definition of structure ****/
    cout << "---- Part 2: Structure definition ----
";
    Student student1; //make structure variable

    /**** Part 3: Accessing structure members ****/
    /**** Part 4: Comparing structure variables ****/
    /**** Part 5: Pointer to structure ****/
    cout << "---- Part 5: Structure data using pointer ----
";
    //Initialize pointer
    Student *stdPtr;
    stdPtr = & student1;
    stdPtr->name;
    stdPtr->student_id;
    stdPtr->total_mark;

    //call pointer print function
    print_structure_using_pointer(*stdPtr);

    /**** Part 6: Nested structure ****/
    cout << "---- Part 6: Nested Structure ----
";
    //print nested structure
    print_nested_structure(student1);
    return 0;
}

void print_structure(Student x) {
    cout << "Student Name: " << x.name << endl;
    cout << "Student ID: " << x.student_id << endl;
    cout << "Total Marks: " << x.total_mark << endl;
}

void print_structure_using_pointer(Student x) {
    cout << "Student Name: " << x.name << endl;
    cout << "Student ID: " << x.student_id << endl;
    cout << "Total Marks: " << x.total_mark << endl;
}

void print_nested_structure(Student x) {
    cout << "Student Address: " << x.pData.student_address << endl;
    cout << "Student City: " << x.pData.student_city << endl;
    cout << "Student State: " << x.pData.student_state << endl;
}
```

Figure 2.13 – Lab 12 Source Code: Structs
CHAPTER III
RELAXNG AND AUTOMATION

RelaxNG was chosen as the grammar language for XML for this research. Although other pattern and query languages have been used for other applications and uses, RelaxNG provided a simpler and more extendable pattern language for creating grammar documents, encapsulated elements, data types, and software developer abstractions.

For a grammar file, there were several common components that were needed to be included. As seen in the RelaxNG figures, the XML syntax for RelaxNG was used for the work presented in this research. A compact version was available, but needed translation before the validation of XML files against it but could be utilized in future work. For the RelaxNG below, the beginning lines contain the grammar tag with defined namespaces. Defining the names for the XML document with ‘xmlns’, the standard RelaxNG namespace was used like any other RelaxNG grammar file. A supplementary annotation namespace, ‘xmlns:a’, provides a RelaxNG standard documentation framework. Finally unique to this research work, ‘ns=http://www.sdml.info/srcML/src’ is the srcML namespace.

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
</grammar>
```
With the annotation namespace shown above, the documentation tags were available to be processed. However, this research did not explore the processing of documentation. Possible future work could explore this feature and provide automated documentation, which may include explanations of why source code is expected to be structured in the specified format and displayed in a webpage. Below is a sample excerpt showing how the documentation is created.

```xml
<a:documentation>
  RELAX NG schema
</a:documentation>
```

Moving further into the grammar file, there is an include statement following a comment. The comments are just as in any XML document. The `start` node is needed for RelaxNG processing like the main function needed by a C++ program. Also, only a single `start` node can be present.

```xml
  <!-- RelaxNG grammar start -->
  <start> … <!—elements --> … </start>
```

As hinted to in the comment within the `start` tag above, `elements` are the next key items. An `element` defines an XML node that is expected within the XML document. To demonstrate an `element`, the examination of the common XML node of a `unit` is shown below with another very common node, `text`.

```xml
  <element name="unit">
    <text/>
  </element>
```

As with any programming languages logic provides far more power and flexibility to create robust applications. The ability of optional elements, a choice between a set of elements, zero or more of a set of elements, and one or more of a set of elements. As seen below in Figure 3.1, these language components are displayed in context of processing simple XML documents.
Furthermore, other language elements needed are attributes, values, and empty nodes. Attributes define the presence of XML node attributes. For example as seen below in Figure 3.2, the ‘comment’ element in srcML has a ‘type’ attribute with that XML node. The value node specifies an exact data type value. In the following case with comments, the value was written to expect the ‘type’ attribute to be ‘block’. Finally, the empty element node just describes the absence of elements or text.
3.1. RelaxNG Definitions and References

Based off these first couple principles of RelaxNG, the simplest file constructed, using.sng, shows these elements in Figure 3.2. For all labs for the Computer Science 1 class examined, all using statements were that as show below in one of the simplest statements in the grammars.

using namespace std;

As parsed by the srcML toolkit, the following line is the marked up XML.

<using>using namespace <name>std</name>;</using>

Then as shown in Figure 3.3, the corresponding RelaxNG was used for validating these such statements.
Although simple, these nodes were the basic building blocks that were used to construct the grammars for the C++ introductory labs. To provide abstractions to implement patterns, definitions and references encapsulate all the previously defined language components. Building off of the comment element in Figure 3.3, the definitions of a ‘comment’ and a ‘header_comment’ in Figure 3.4, which was contained within the file comments.rng, demonstrate an abstraction of some possibly permutations of comments found in C++. With ‘header_comment’, the type of comment is specified and forces the RelaxNG processing to only allow for a srcML ‘comment’ to contain the attribute ‘block’. With the first definition titled ‘comment’, this abstraction when referred will have the flexibility of any attribute text for ‘type’.
To continue demonstrating specifying expected text values, the include statements shown in Figure 3.5 are defined exactly in the grammars in Figure 3.6. Also seen in Figure 3.5, the specific use of the cpp namespace is needed for the srcML elements nodes include, directive, and file. This concept is a feature of srcML.
Figure 3.5 Include Statement in srcML
Figure 3.6 Includes Definition RelaxNG
The next definition file, statements.rng, contains the bulk of the grammar referenced since declarative statements and expression statements make up a majority of C++ programs. The first statement shown in Figure 3.7 is the return element definition. Although simple in and of itself, this reference, ‘ref’, refers to a definition elsewhere defined as ‘expression’, which is seen in Figure 3.8.

```
<define name="return">
  <element name="return">
    <text/> <!-- return -->
    <ref name="expression"/>
    <text/> <!-- ; -->
  </element>
</define>
```

Figure 3.7 Return Definition RelaxNG

Defining the reference to ‘expression’ in Figure 3.7, Figure 3.8 shows the definition for ‘expression’ that is initiated by the srcML element expr. Typically initiated by the ‘expression_statement’ seen below, the ‘expression’ was also found in definitions like the ‘return’ above.

```
<define name="expression_statement">
  <element name="expr_stmt">
    <ref name="expression"/>
  </element>
</define>
```

As seen in the comments within Figure 3.8, a couple of expressions were of particular note that caused for added complexity of the ‘expression’ definition. Specifically, array based expressions and pointer deletions were these more complex expressions. Also, definitions like ‘function_call’ will be described later on.
<define name="expression">
  <element name="expr">
    <zeroOrMore>
      <optional><text/></optional>
      <choice>
        <!-- delete[] ptr -->
        <element name="index">
          <text/>
        </element>
        <element name="name">
          <oneOrMore>
            <choice>
              <text/>
              <element name="name">
                <text/>
              </element>
              <element name="index">
                <text/>
                <element name="expr">
                  <choice>
                    <!-- delete[] ptr -->
                    <element name="index">
                      <text/>
                    </element>
                    <element name="name">
                      <oneOrMore>
                        <choice>
                          <text/>
                          <element name="name">
                            <text/>
                          </element>
                          <element name="index">
                            <text/>
                            <element name="expr">
                              <choice>
                                <element name="name">
                                  <text/>
                                </element>
                              </choice>
                            </element>
                          </element>
                        </choice>
                      </oneOrMore>
                    </element>
                    <element name="name">
                      <oneOrMore>
                        <choice>
                          <text/>
                          <element name="name">
                            <text/>
                          </element>
                        </choice>
                      </oneOrMore>
                    </element>
                  </choice>
                </element>
              </choice>
            </oneOrMore>
          </choice>
        </element>
      </choice>
    </zeroOrMore>
    <zeroOrMore>
      <choice>
        <ref name="function_call"/>
        <text/>
        <element name="name">
          <text/>
        </element>
      </choice>
    </zeroOrMore>
  </element>
</define>
Similarly, the ‘declarative_statement’ used the same pattern of separating the statement from the ‘declaration’. In Figure 3.9, the ‘declarative_statement’ is shown.

Even lengthier, the ‘declaration’ definition is shown in Figure 3.10. Noted within a couple comments like the ‘expression’ definition, there are elements of code that cause the grammar to facilitate complex situations. Pointers are denoted in this figure just as text elements. As seen in the comments and includes, this simple text element could be constrained for the ‘*’ as the string data value. Although understood by the pointer syntax being the only situation to cause this text element at this point of research into the
source grammar, the validation of the pointer notation could be future work to assist in validating the srcML output itself.

```xml
<define name="declaration">
  <element name="decl">
    <element name="type">
      <element name="name">
        <text/>
      </element>
      <optional><text/></optional>
    </element>
    <optional>
      <element name="name">
        <text/>
      </element>
    </optional>
    <!-- Pointer, * -->
    <optional><text/></optional>
    <optional>
      <element name="name">
        <text/>
      </element>
    </optional>
    <!-- double numbers[ARRAYLEN] = ...; 'numbers' is name element here -->
    <optional>
      <element name="name">
        <text/>
      </element>
      <!-- More than 1 for multidimensional arrays -->
      <zeroOrMore>
        <!-- <index>[<expr><name>ARRA YLEN</name></expr>]</index> -->
        <element name="index">
          <text/>
        </element>
        <!-- could just be a literal -->
        <optional>
          <element name="expr">
            <choice>
              <text/>
            </choice>
          </element>
        </optional>
        <!-- double numbers[ARRAYLEN] = ...; 'numbers' is name element here -->
        <optional>
          <element name="name">
            <text/>
          </element>
        </optional>
        <!-- More than 1 for multidimensional arrays -->
        <zeroOrMore>
          <!-- <index>[<expr><name>ARRA YLEN</name></expr>]</index> -->
          <element name="index">
            <text/>
          </element>
        </optional>
      </optional>
    </optional>
  </element>
</define>
```
Array declaration with values.
double numbers[ARRAYLEN] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};
Aligned with what the students learned in the lab sequence, conditionals added logic and control. Contained within the conditionals.rng file, *if* and *switch* definitions were defined and abstracted out. Seen in Figure 3.11, the *if* definition appears much cleaner and simpler as far as typical coding styles. Other than the *if, then,* and *else* elements, most of this definition was defined using references and text elements. The structure of this definition shows the evolution of the source grammars into what may resemble the framework going forward for developers to utilize in a simpler fashion.

Figure 3.10 Declaration Definition RelaxNG
As referenced in the *if* definition, the *ifblock* also resides in the conditions.rng file. Later defined, the *switch* statement is also referenced in the block of the *if*. In the argument of coding style, the choice between the ‘block’ element and an ‘expression_statement’ simply shows the presence of curly braces or not following, respectively, the if condition.
Completing the conditionals, the switch statement contains multiple parts including the condition, case statement, and default statement. Down to the details, these components contain mostly expression statements, text, and a break statement.
Figure 3.13 Switch Definition RelaxNG

Progressing onto more complex topics within C++ as in the experience of an introductory student, loops provide the control structure to better manipulate data. Within what was created for the included files, the loops.rng file contained the while, for, and do while loops.
Even though each type of loop can be modified to perform like another, the structure and key words vary between each one. First, the while loop is defined in Figure 3.14.

```
<define name="while">
  <element name="while">
    <text/>
    <element name="condition">
      <text/>
      <ref name="expression"/>
      <text/>
    </element>
    <text/>
    <element name="block">
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="declarative_statement"/>
          <ref name="expression_statement"/>
          <ref name="if"/>
          <ref name="switch"/>
          <ref name="for"/>
          <ref name="while"/>
        </choice>
      </oneOrMore>
    </element>
  </element>
</define>

Figure 3.14 While Loop Definition RelaxNG
```

As seen in Figure 3.14, the while loop contains very simple parts once the previously defined patterns are established. The key word ‘while’ in the root element begins the loop followed by text, a whitespace, a ‘condition’ containing an ‘expression’ surrounded by whitespace and parenthesis, and a block with many references possible to possibly occurring. The for loop offers more elements to occur within its srcML translation, which corresponds to the syntax of the language. Instead of just the condition after the keyword, the for loop has the ‘init’ element of a ‘declaration or an ‘expression’, a ‘condition’ like the while’s ‘condition’, and an ‘incr’ element that increments the variable. Seen in the
comment in Figure 3.15 within this section, the increment may increment a variable by a value of another variable instead of just a literal, ‘i += step’.

```xml
<define name="for">  
   <element name="for">  
      <text/>
      <element name="init">  
         <choice>
            <ref name="declaration"/>
            <ref name="expression"/>
         </choice>
         <text/>
      </element>  
      <element name="condition">  
          <element name="expr">  
             <element name="name">  
                 <text/>
             </element>
             <text/>
             <element name="name">  
                 <text/>
                 <optional>  
                     <text/>
                     <element name="name">  
                         <text/>
                     </element>
                 </optional>  
             </element>
             <optional>  
                 <text/>
                 <element name="name">  
                     <text/>
                 </element>
             </optional>  
          </element>
          <text/>
          <element name="incr">  
             <element name="expr">  
                 <optional><text/></optional>
                 <element name="name">  
                     <text/>
                 </element>  
                 <optional><text/></optional>
             </element>
          </optional>
          <text/>
      </element>  
      <text/>
   </element>  
</define>
```

`for (int i=m;i<n; i += step) ... ‘i += step’`
Figure 3.15 For Loop Definition RelaxNG

Although more rare, the do while loop occurs once within the first 12 labs. In Figure 3.16, the do while that was used is shown translated using srcML.

```
<do>
   <block>
      <expr_stmt>
         <expr>
            <name>num</name> =
            <call>
               <name>input_an_int</name>
               <argument_list>()</argument_list>
            </call>
         </expr>
      </expr_stmt>
      <expr_stmt>
         <expr>
            <name>sum</name> +=
            <name>num</name>
         </expr>
      </expr_stmt>
   </block>
   <while>
      <condition>
         <expr>
            <name>num</name> != 0
         </expr>
      </condition>
   </while>
</do>
```

Figure 3.16 Do While Loop srcML

Since we only had to validate the grammar against this one occurrence of the do while loop, the possibly permutations of the do while could not have been discovered through this research. Therefore, the RelaxNG in Figure 3.17 was created to validate a common
structure of a do while loop. Possible permutations may include whitespace modifications, more elements used within the loop block, and nesting loops within.

Figure 3.17 Do While Loop Definition RelaxNG

The final included file with elements defined was the functions.rng file, which included definitions for function calls, function declarations, and function definitions. For the first couple of labs even though students didn’t write any functions other than the main function, function calls were made to libraries like input, output, and math. Figure 3.18 shows the definition of the ‘function_call’.
Next, the function declarations were used for function prototypes before the main function in the labs. Figure 3.19 shows the definition of the ‘function_declaration’.
Finally, the function definition provided a generic layout of a function within the programs. As seen later, some functions are defined separately to precisely define an order of what elements were expected as opposed to a generic implementation. This concept will allow for future work to vary strictness of validation for different parts being observed at a time. Figure 3.20 shows the generic implementation of a ‘function’ definition.
Figure 3.20 Function Definition RelaxNG
To summarize the development of the RelaxNG files and elements, there were many standalone definitions that needed to be included together. The result of this need produced the cpp-core.rng file. Instead of including what files were needed from file to file already defined, one file was decided upon to be used. Since there was overlap between all the files already and RelaxNG seems to simply combine these files into one at runtime, each definition could access every reference when the cpp-core.rng file combined these. Figure 3.21 shows the core include file.

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src"
  >

  <a:documentation>
  RELAX NG schema for core includes

  Brandon Sedgwick
  bms49@zips.uakron.edu
  2013
  </a:documentation>

  <include href="comments.rng"/>
  <include href="includes.rng"/>
  <include href="usings.rng"/>
  <include href="statements.rng"/>
  <include href="functions.rng"/>
  <include href="conditionals.rng"/>
  <include href="loops.rng"/>
</grammar>
```

Figure 3.21 C++ Core RelaxNG Include Grammar File
3.3. Application of Validation Using Grammars

Building up from the prior sections demonstrating a complete RelaxNG file composition and the abstracted out definitions, validation was automated in order to proceed with integrating with the lab generation sequence. First, the library being used was libxml. However efficient this standard library was initially, there were a couple of miniscule errors that continued to occur while validating even the simplest of srcML files. Most notably, the libxml library raised errors for incomplete elements that seemed to be perfectly complete in XML form. Experimenting with other libraries, the seemingly most accurate library was from Thai Open Source Software, jing, which is a RelaxNG validator written in Java\(^6\). The jar file provided on their web page was needed to validate the srcML against the RelaxNG files.

As seen in Figures 3.2, 3.3, and 3.4 of this section, the complete RelaxNG grammar implemented for just Lab 1 encompasses multiple files. However, Figure 3.3 and Figure 3.4 are reused for every lab throughout. To run the validation tool, the following command line scripts were executed. The first line translates the Lab1 source code into srcML XML. Then the second line validates the srcML output with the Lab1 RelaxNG grammar file. Finally, the output from the validation would be stored in Lab1Results.txt.

\[
\text{src2srcml -q Lab1.cpp -o Lab1.srcml.xml} \\
\text{java -jar jing.jar Lab1.rng Lab1.srcml.xml > Lab1Results.txt}
\]

Using these guidelines of files and command line scripts with more robust code, the experimentation was able to be carried out, which is described in more detail in Chapter 4.

All testing was executed on a Mac OS X and Ubuntu (10.04+ Operating System versions).

Utilization of these scripts allowed for automation.
The experiment conducted consisted of four main data parts: source code from students, srcML translation, RelaxNG grammar files as described in Chapter 3, and results from the automation script. Since unique grammars rely on building up from individual elements, on a lab to lab basis unit testing individual elements separately created a framework to be used at a large scale, complete program analysis. As seen in Figure 4.1, the complete automation process is illustrated from when the students made code changes to the use of the validation script to output the results.
As reported by Zimmerman et. al.[11], the effort to grade the internal structure of their submitted source code is the most time consuming and difficult. As with traditional testing, validation, and verification, simply running the students’ programs against sample test data will create concrete results that can be measured and checked programmatically. However, as Zimmerman et. al. shows us, the difficulty for grading is hindered by the human element needed to review and mark up of source code files. What is presented in this paper can be a solution to the internal structure of source code grading during the assignment completion rather than just after submission. With the addition of using version control, each iterative commit can be treated as a submission in this scenario and be auto-graded for instant feedback.
4.1. Complete lab files of srcML with RelaxNG

For a complete program in C++, there are just a couple of simple parts. The first expected element is a header comment followed by includes. Then the main function with comments, declarations, and expressions completes the remaining elements of the source code file. At any point in or out of the main function, text like spaces, tabs, and new lines may be found.

For example, taking the sample first lab for the introductory computer science course as seen in Figure 1 the srcML can easily be derived and explained. Immediately following is the simple C++ code for the source file.

```cpp
/*
 A header comment.
 */
#include <iostream>
using namespace std;
int main() {
  // Prints welcome message...
  cout << "Welcome to Lab 1." << endl;
  return 0;
}
```

Translated using the srcML toolkit, the resulting XML can be used for validation against RelaxNG files. Figure 4.2 displays the first couple of elements of the RelaxNG file for the lab shown above. All labs regardless of content would read these first couple of elements to simply describe the file itself.
As seen in Figure 4.2, the srcML result of the C++ source code from Lab 1 seen in Chapter 2 was translated into XML. The corresponding RelaxNG grammar file to validate its basic structure and makeup expected, Figure 4.3 shows some conclusion and summation of the work done for the included definitions described in Chapter 3. The “unit” element contains two attributes of language, which is C++ in this case, and filename, which is just the source code file name passed in to the srcML toolkit. The first
couple lines of the source code defines the pattern of header comment, iostream include, and a ‘using namespace std’ statement. Each of these three elements is references to a definition elsewhere. At this point, the structure of this function as compared to the above example looks for a type that is \textit{int}, a text value that is simply a space, a name element with text that is \textit{main}, a parameter list that is empty, and the beginning of the function block with text, which consists of a curly brace, spaces, and a newline. The elements following the beginning of the function block shows what is expected in each of the students' main function. Wrapped around the \textit{oneOrMore} and \textit{choice} elements allows for any combination of statements, comments, and text, which is typically just spaces, newlines, and tabs.

For Lab 2 seen in Figure 2.2, most of the structure of the source code is exactly the same as Lab 1. An added element was the \textit{function_call} as noted by the call element node in srcML in Figure A.2. With Lab 3 that was described in Chapter 2, most of the main RelaxNG structure is very similar, if not identical, to Lab 2’s RelaxNG. As seen in Figure A.4, there were many more expression and declaration statements used in Lab 3.

For Lab 5, Figure A.8 shows what was described in Chapter 2. For the code structure of Lab more complex if then statements were combined with switch statements. The validating RelaxNG grammar file for Lab 5 added more references like \textit{if} and \textit{switch}.

In Figure A.10, Lab 6’s source code is translated using the srcML toolkit for the resulting XML that was used. With the added elements for loops, references to \textit{switch}, \textit{for}, and \textit{do} are seen in Figure A.11.

Lab 9 contained the first occurrence of a variable declaration outside the main function. Also, the function declarations were limited to a single declaration, which occurred
before the variable declaration. As seen below in Figure A.17, the lines in the RelaxNG
beginning the grammar before the function definition for the main function describe those
restrictions.

Lab 10’s RelaxNG in Figure 4.21 shows a perfect example of forcing more structure to
a source code file. With the XML comments as guides, the incremental parts in the lab
are shown from part 0 through part 4. Although just generic functions were created for
the initial research, specific functions can be created in the RelaxNG definitions. A
couple enforcments in the validation may include forcing a set return type, function name
pattern or exact name, number of parameters, parameter types, number of statements,
types of statements, and placement of comments.

In Lab 11, as shown in Figure A.20 its srcML, the binary search and selection sort
algorithms were implemented. To set this lab up, the first two functions were required by
students to complete them while the main function acted as a test driven development
function for them to follow along with test cases.

As seen within the srcML for Lab 12 in Figure 4.24, pointers were the main focus.
Although a complex topic in C++ for people to learn, the RelaxNG grammar to validate
expressions with pointers did not add much complexity at all. With arrays already
accounted for, mostly just added text value of the pointer notation, ‘*’, was needed to be
added.
4.2. Experiment Design
From over 300 students' labs of a Computer Science 1 course over 3 semesters, C++ source code was examined and compared against the source code validation. Each lab had its own unique set of elements that was expected in a final solution of the C++ source code. As described in detail from Lab 1 through Lab 12 in Chapter 2 the source code expected for each lab was explained. For this paper, all twelve labs were completely examined and validated for the 335 students. However, a caveat is that labs were shifted around after the second semester of the sequence. The labs on structs and classes were moved to the second computer science course.

Once the lab’s source code has been committed to the repository, it is ready to be graded and reported on back to the students. The course repository's directory structure as seen by instructors and graders contains each class section with folders for each student. Each student is given folders for labs, projects, and evaluations. These labs folders contain the student submission for each lab, which is what is analyzed by this research. The evaluations folder matches the working directory of the student which is written to by the instructors and graders and read by the student.

4.3. Experiment Analysis and Results
Most students, as expected, completed the labs similarly by having similar results when tested against the validation grammars. Out of the sample set of 362 sample students, the fully tested results focused on Lab 1 through Lab 4. For sample output that could be parsed and placed into the lab report file, see below.
In Figure 4.26, the output for validating Lab 10 is shown. Since perfectly valid source code would produce no output, this output shows some errors. The first error, the source code was missing any valid expression like block, expr_stmt, if, or return. Then the second error shows using indexing incorrectly or out of place.

For Lab 1, nearly all processed labs yielded good results. For the first round of testing, 94% of all the students passed the source code validation. With closer observation of the results, more flexibility was added in the validation of the source file. A couple of the source code files were placing comments in various places like in between include statements. After revisions the RelaxNG file for Lab 1 and running through the sample data again, the flexibility in comment placement changed to validated labs to over 98% of the source code files. With further analysis of the validation errors, the most common errors in Lab 1 source code files was the absence of a header comment. Since Lab 1 required a header comment, the validation tool worked successfully to discover a missing core element relative to the lab.

Similar to Lab1, Lab2 results showed a 94% successful validation rate for the given source code files. A large chunk of the 6% labs that failed to validate included a missing header comment. Just one source code file failed to include the return statement in the main function.
Lab 3 and 4 results were even less successful at being completely valid for the expected source code grammars. Lab 3 results showed only a 80% total successful validation rate for the given source code files, while Lab 4 had only a 63% complete validation success. Keep in mind, this decreasing percentage of total source code validation consisted of mostly one or two validation infractions per student. The total percentage represents the number of students with completely valid source code to the total set of students. With more include statements that were being used as the students learned to use more libraries, the greater the chance extra libraries could have been included. One key grammar decision for validation was checking for specific included library names. However, only three of the over 300 lab source files in Lab 4 had one extra include file not specified by the instructions.

<table>
<thead>
<tr>
<th>Lab</th>
<th>Students</th>
<th>KLOC</th>
<th>Time (sec)</th>
<th>LOC/sec</th>
<th>Students' lab/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>362</td>
<td>7.5</td>
<td>99</td>
<td>75</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>362</td>
<td>11.5</td>
<td>99</td>
<td>116</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>362</td>
<td>21.6</td>
<td>131</td>
<td>164</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>362</td>
<td>37.9</td>
<td>140</td>
<td>270</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>362</td>
<td>54.3</td>
<td>141</td>
<td>385</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>362</td>
<td>38.8</td>
<td>137</td>
<td>283</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>362</td>
<td>30.7</td>
<td>128</td>
<td>239</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>362</td>
<td>32.6</td>
<td>133</td>
<td>245</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>362</td>
<td>38.1</td>
<td>137</td>
<td>278</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>362</td>
<td>40.3</td>
<td>134</td>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>362</td>
<td>41.9</td>
<td>132</td>
<td>317</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>133</td>
<td>14.2</td>
<td>113</td>
<td>125</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.4 Source Grammar Validation Run Times
CHAPTER V

FUTURE WORK

Moving forward the research direction can apply to the areas of software engineering education and source code grammars to enforce standards or practices. Applications of this research can use many approaches and tools. Similar to the solution proposed by Heckman's Automated Warning Application for Reliability Engineering, AWARE, tool to provide static analysis alerts instantly to software engineers, possibly developing ranking and filtering system of the results of the source code grammar validation can provide a more detailed and helpful reasoning of code issues to the student or developers. When expanding to larger projects and a greater number of files validating at a time, a ranking and filtering feature could prove very useful, as how the AWARE tool approached type accuracy, code locality, and generated test failure factors [5].

5.1. Software Engineering Education

In teaching courses for software engineering, adoption of agile processes of continuous iteration and feedback will exponentially accelerate the growth and strength of writing code ability in students, coinciding with concepts to drive students to be thinking and working iteratively in an agile manner as opposed to the waterfall approach. Moving away from waterfall in developing projects in school or projects in industry may prove to assist in cooperative working and fighting off procrastination, as Macek and Komarek
concluded by students' responses that they were more motivated with iteratively working on projects[7].

Since typically source code feedback from an instructor or grader is done after the assignment is completed and can take long amounts of time, automated feedback on source code provides instant feedback to the student. When a software engineering class can typically contain around 20 to 30 students, classes could no longer be restrained by the inability to provide timely feedback to students. Classes, like introductory programming classes, could easily expand to larger sizes of 100 to 200 students.

5.2. Source Code Grammars to Enforce Standards

Similar to academic settings for software engineering, automated tools for evaluation of source code can be applied to industry software development. For a concrete example of what could be done in industry, working at Diebold as a software architect provided me with the opportunity to examine all the source code for the cloud applications and services. Within each commit to the code base for each project a commit hook, which is a script that runs when a commit event is triggered by a call into the repository, could run a tool against the changed source code structures to simulate a similar process to that shown in this paper for validating labs' source code. A series of validations could occur at this step to check for varying levels of validation of allowable code format, structure, and specifics like library or API usage.

5.3. Commit by Commit Differences Validation

In addition to validating the final submitted version of the source code, validating individual commits of source code by the difference in source code can narrow down what is expected at that point. For example, in a lab there may be multiple steps to
complete the lab. At each step there will be necessary changes made to the source code, such as adding a function definition for a provided function declaration as a prototype. Using a technique like a version control commit hook possibly looking for a keyword to trigger the tool to validate and provide feedback, each step completed can provided the student instant grading of their source code structure and elements being edited.
CHAPTER VI
CONCLUSION

An issue with software engineering education is relying on the human element for source code feedback. With automated tools to provide instant evaluation of the source code structure and elements, students can learn through iterations of writing source code for projects, labs, and other assignments. As shown in this paper, there is endless future work to be built upon automated grading, which includes validating internal source code structures as with traditional testing. Future work holds promise to delivering agile software processes to students and instant feedback for iterative learning processes.


APPENDIX

EXPERIMENT FILES

Figure A.1 Lab 1 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  ns="http://www.sdml.info/srcML/src" >
  <!-- get definitions of components-->
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>
      <oneOrMore>
        <ref name="comment" />
      </oneOrMore>
      <optional><text/></optional>
      <ref name="iostream" />
      <optional><text/></optional>
      <ref name="using" />
      <optional><text/></optional>
      <optional><ref name="comment"/></optional>
      <element name="function">
        <element name="type">
          <element name="name">
            <text/>
          </element>
        </element>
        <text/>
        <element name="parameter_list">
          <text/>
        </element>
        <text/>
        <element name="block">
          <choice>
            ...
          </choice>
        </element>
      </element>
    </element>
  </start>
</grammar>
```
Figure A.2 Lab 2 srcML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.sdml.info/srcML/src"
     xmlns:cpp="http://www.sdml.info/srcML/cpp" language="C++" filename="complete-2.cpp">
    <comment type="block">
        /*
        complete-2.cpp
        Calculate the time it will take to drive given a
distance in miles and a speed in miles per hour (mph)
*/
    </comment>
    <cpp:include>
        #<cpp:directive>include</cpp:directive>
        <cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
    <using>using namespace <name>std</name>;</using>
    <function>
        <type>
            <name>int</name>
        </type>
        <name>main</name>
        <parameter_list>()</parameter_list>
        <block>{
            <comment type="line">// variables to hold distance and speed</comment>
            <decl_stmt><decl><type><name>double</name></type><name>distance1</name>
                =<init><expr>100</expr></init></decl>;</decl_stmt>
            <decl_stmt><decl><type><name>double</name></type><name>distance2</name>
                =<init><expr>50</expr></init></decl>;</decl_stmt>
            <decl_stmt><decl><type><name>double</name></type><name>speed</name>
                =<init><expr>35</expr></init></decl>;</decl_stmt>
            <comment type="line">// input the distances in miles</comment>
            <expr_stmt><expr><name>cout</name> &lt;&lt; "Enter two integer distances (in
miles):" &lt;&lt; endl</expr>;</expr_stmt>
            <expr_stmt><expr><name>cin</name> &gt;&gt; <name>distance1</name> &gt;&gt;
                <name>distance2</name></expr>;</expr_stmt>
            <comment type="line">// input a speed in mph</comment>
            <expr_stmt><expr><name>cout</name> &lt;&lt; "Enter an integer for the travel
speed (in mph):" &lt;&lt; endl</expr>;</expr_stmt>
            <expr_stmt><expr><name>cin</name> &gt;&gt; <name>speed</name></expr>;</expr_stmt>
            <comment type="line">// variable to hold resulting time</comment>
            <decl_stmt><decl><type><name>double</name></type><name>travelTime</name>
                =<init><expr>3</expr></init></decl>;</decl_stmt>
```

76
// calculate the drive time based on distance and speed
<expr_stmt><expr><name>travelTime</name> = (<name>distance1</name> + <name>distance2</name>) / <name>speed</name></expr>; </expr_stmt>

// output the drive time
<expr_stmt><expr><name>cout</name> &lt;&lt; "The travel time is " &lt;&lt; <name>travelTime</name> &lt;&lt; " hours." &lt;&lt; <name>endl</name></expr>; </expr_stmt>
<return>return <expr>0</expr>; </return>
Figure A.3 Lab 2 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
         xmlns:ns="http://www.sdml.info/srcML/src">
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"></optional>
      <optional><attribute name="filename"></optional>
      <oneOrMore>
        <ref name="comment" />
      </oneOrMore>
      <optional><text></optional>
      <ref name="iostream" />
      <optional><text></optional>
      <ref name="using" />
      <optional><text></optional>
      <optional><ref name="comment"></ref>
      <element name="function">
        <element name="type">
          <element name="name">
            <text/>
          </element>
        </element>
        <text/>
        <element name="name">
          <text/>
        </element>
        <element name="parameter_list">
          <text/>
        </element>
        <text/>
        <element name="block">
          <oneOrMore>
            <choice>
              <text/>
              <ref name="declarative_statement"/>
              <ref name="expression_statement"/>
              <ref name="comment"/>
            </choice>
          </oneOrMore>
        </element>
      </element>
      <optional><ref name="comment"></ref>
      <element name="return">
        <optional>
        </optional>
      </element>
    </element>
  </start>
</grammar>
```
<oneOrMore>
  <choice>
    <text/>
    <ref name="expression"/>
  </choice>
  <ref name="expression_statement"/>
</oneOrMore>
Figure A.4 Lab 3 srcML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.sdml.info/srcML/src"
     xmlns:cpp="http://www.sdml.info/srcML/cpp" language="C++" filename="complete-3.cpp">
<comment type="block">
  ***********************************************************
  ***********
  * complete-3.cpp
  * Purpose: Demonstration of C++ library headers
  * This program will display (on the console) a table containing 4 usernames along with a score for each.

  ***********************************************************</comment>

<cpp:include>
  #<cpp:directive>include</cpp:directive>
  <cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<cpp:include>
  #<cpp:directive>include</cpp:directive>
  <cpp:file>&lt;iomanip&gt;</cpp:file></cpp:include>
<cpp:include>
  #<cpp:directive>include</cpp:directive>
  <cpp:file>&lt;cmath&gt;</cpp:file></cpp:include>
<cpp:include>
  #<cpp:directive>include</cpp:directive>
  <cpp:file>&lt;string&gt;</cpp:file></cpp:include>
<using>
  using namespace <name>std</name>;</using>

<function>
  <type><name>int</name></type>
  <name>main</name><parameter_list>()</parameter_list> <block>{
    <comment type="line">// four scores for four students</comment>
    <decl_stmt>
      <decl><type><name>int</name></type> <name>score1</name> = <init>23</init></decl>
    </decl_stmt>
    <decl_stmt>
      <decl><type><name>int</name></type> <name>score2</name> = <init>54</init></decl>
    </decl_stmt>
    <decl_stmt>
      <decl><type><name>int</name></type> <name>score3</name> = <init>7</init></decl>
    </decl_stmt>
    <decl_stmt>
      <decl><type><name>int</name></type> <name>score4</name> = <init>211</init></decl>
    </decl_stmt>
    <decl_stmt>
      <decl><type><name>string</name></type> <name>person1</name> = <init>"Ava"</init></decl>
  }<block>
</function>
```
```cpp
<decl_stmt><decl><type><name>string</name></type> <name>person2</name> =<init> <expr>"Tom"</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>string</name></type> <name>person3</name> =<init> <expr>"David"</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>string</name></type> <name>person4</name> =<init> <expr>"Kathy"</expr></init></decl>;</decl_stmt>

<expr_stmt><expr><name>cout</name> &lt;&lt; "Please type four persons' name.
"</expr>;</expr_stmt>
<expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>, <argument><expr><name>person1</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>, <argument><expr><name>person2</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>, <argument><expr><name>person3</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>, <argument><expr><name>person4</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>

<comment type="line">// user inputs all four scores</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Please type four (integer) scores
"</expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name> &gt;&gt; <name>score1</name> &gt;&gt; <name>score2</name> &gt;&gt; <name>score3</name> &gt;&gt; <name>score4</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argument><expr>20000</expr></argument>, <argument><expr>'\n'</expr></argument>)</argument_list></call></expr>;</expr_stmt>
```

81
// calculate the average score
<decl_stmt><decl><type><name>double</name></type> <name>average</name> =<init> <expr><call>(<name>double</name>)<argument_list>(<argument><expr><name>score1</name> + <name>score2</name> + <name>score3</name> + <name>score4</name></expr></argument>)</argument_list></call> / 4</expr></init></decl>;
</decl_stmt>

// output a table of scores
<expr_stmt><expr><name>cout</name> &lt;&lt; <call><name>setw</name><argument_list>(<argument><expr>22</expr></argument>)</call> &lt;&lt; <name>left</name> &lt;&lt; "PERSON" &lt;&lt; <call><name>setw</name><argument_list>(<argument><expr>22</expr></argument>)</call> &lt;&lt; <name>right</name> &lt;&lt; <name>score1</name> &lt;&lt; <call><name>setw</name><argument_list>(<argument><expr>22</expr></argument>)</call> &lt;&lt; <call><name>abs</name><argument_list>(<argument><expr><name>score1</name> - <name>average</name></expr></argument>)</call> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; <call><name>setw</name><argument_list>(<argument><expr>22</expr></argument>)</call> &lt;&lt; <name>left</name> &lt;&lt; <name>person2</name> &lt;&lt; <name>score2</name> &lt;&lt; <call><name>setw</name><argument_list>(<argument><expr>22</expr></argument>)</call> &lt;&lt; <call><name>abs</name><argument_list>(<argument><expr><name>score2</name> - <name>average</name></expr></argument>)</call> &lt;&lt; <name>endl</name></expr>;</expr_stmt>

cout << setw(22) << left << person3 << " 
" << setw(22) << right << score3 << " 
" << setw(22) << abs(score3 - average) << " 
" << endl;

cout << setw(22) << left << person4 << " 
" << setw(22) << right << score4 << " 
" << setw(22) << abs(score4 - average) << " 
" << endl;

return 0;
Figure A.5 Lab 3 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
         ns="http://www.sdml.info/srcML/src">
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
        <optional><text/></optional>
        <ref name="using"/>
        <optional><text/></optional>
        <optional><ref name="comment"/></optional>
      </oneOrMore>
    </element>
    <element name="function">
      <element name="type">
        <element name="name">
          <text/>
        </element>
      </element>
      <text/>
      <element name="name">
        <text/>
      </element>
      <element name="parameter_list">
        <text/>
      </element>
      <element name="block">
        <oneOrMore>
          <choice>
            <text/>
            <ref name="declarative_statement"/>
            <ref name="expression_statement"/>
            <ref name="comment"/>
          </choice>
        </oneOrMore>
      </element>
    </element>
  </start>
</grammar>
```
Figure A.6 Lab 4 srcML

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
language="C++" filename="complete-4.cpp"><comment
type="block">**************************************************************************
**
* complete-4.xpp
* Purpose: Demonstrate "if" statements
**************************************************************************</comment>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;iomanip&gt;</cpp:file></cpp:include>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;string&gt;</cpp:file></cpp:include>
<using>using namespace <name>std</name>;</using>
<function><type><name>int</name></type><name>main</name><parameter_list>()</parameter_list> <block>{
  <comment type="line">// Constant symbols to use throughout the program.</comment>
  <decl_stmt><decl><type><name>const</name> <name>int</name></type> <name>SUN</name> =<init> <expr>1</expr></init>, <name>MON</name> =<init> <expr>2</expr></init>, <name>TUE</name> =<init> <expr>3</expr></init>,
    <name>WED</name> =<init> <expr>4</expr></init>, <name>THU</name> =<init> <expr>5</expr></init>, <name>FRI</name> =<init> <expr>6</expr></init>,
    <name>SAT</name> =<init> <expr>7</expr></init></decl>;</decl_stmt>
  <decl_stmt><decl><type><name>const</name> <name>int</name></type> <name>JAN</name> =<init> <expr>1</expr></init>, <name>FEB</name> =<init> <expr>2</expr></init>, <name>MAR</name> =<init> <expr>3</expr></init>,
    <name>APR</name> =<init> <expr>4</expr></init>, <name>MAY</name> =<init> <expr>5</expr></init>, <name>JUN</name> =<init> <expr>6</expr></init>,
    <name>JUL</name> =<init> <expr>7</expr></init>, <name>SEP</name> =<init> <expr>8</expr></init>, <name>SEP</name> =<init> <expr>9</expr></init>, <name>OCT</name> =<init> <expr>10</expr></init>, <name>NOV</name> =<init> <expr>11</expr></init>, <name>DEC</name> =<init> <expr>12</expr></init></decl>;
  <comment type="line">// variables to hold input.</comment>
  <decl_stmt><decl><type><name>int</name></type> <name>month</name>, <name>day</name></decl>;</decl_stmt>
  <decl_stmt><decl><type><name>string</name></type> <name>person</name></decl>;</decl_stmt>
  <comment type="line">// prompt user to enter a name</comment>
  <expr_stmt><expr><name>cout</name> &lt;&lt; "What is your name?
    &gt;"</expr>;</expr_stmt>
  <expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>)</call></expr>;</expr_stmt>
  <comment type="line">// prompt user to enter a day of the week</comment>
  <expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>)</call></expr>;</expr_stmt>
  <comment type="line">// prompt user to enter a month of the year</comment>
  <expr_stmt><expr><call><name>getline</name><argument_list>(<argument><expr><name>cin</name></expr></argument>)</call></expr>;</expr_stmt>
}</function>
<argument><expr><name>person</name></expr></argument>)</argument_list></call></expr>;

<comment type="line">// prompt user to enter month</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Type a number " &lt;&lt; 
&lt;&lt; " to select a month:
&gt;
<expr_stmt><expr><name>cin</name> &gt;&gt; <name>month</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argume
t><expr>20000</expr></argument>,
<argument><expr>'
'</expr></argument>)</argument_list></call></expr>;</expr_stmt>

<comment type="line">// prompt user to enter day of the week</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Type a number " &lt;&lt; 
&lt;&lt; " to select a day (Sunday - Saturday)

&gt;"
<expr_stmt><expr><name>cin</name> &gt;&gt; <name>day</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argume
nt><expr>20000</expr></argument>,
<argument><expr>'
'</expr></argument>)</argument_list></call></expr>;</expr_stmt>

<expr_stmt><expr><name>cout</name> &lt;&lt; "You chose month: " &lt;&lt; 
<name>month</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>

<expr_stmt><expr><name>cout</name> &lt;&lt; "You chose day: " &lt;&lt; 
<name>day</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>

<comment type="block">/*
* Part 1: compound conditional
*/</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "\n----- part 1: ----- " &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>
<if>if<condition>(<expr><name>day</name> == <name>SUN</name> || <name>day</name> == <name>SAT</name>)</condition><then>
<block>
<expr_stmt><expr><name>cout</name> &lt;&lt; "It is a weekend!" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>
</block></then></if>

<comment type="block">/* YOUR CODE HERE */</comment>
<if>if<condition>(<expr><name>month</name> &gt;= <name>JAN</name> &amp;&amp; <name>month</name> &lt;= <name>APR</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Spring semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else <if>if<condition>(<expr><name>month</name> &gt;= <name>MAY</name> &amp;&amp; <name>month</name> &lt;= <name>AUG</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Summer semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else <if>if<condition>(<expr><name>month</name> &gt;= <name>SEP</name> &amp;&amp; <name>month</name> &lt;= <name>DEC</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Autumn semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else <if>if<condition>(<expr><name>month</name> &gt;= <name>SEP</name> &amp;&amp; <name>month</name> &lt;= <name>NOV</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Fall semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else <if>if<condition>(<expr><name>month</name> &gt;= <name>NOV</name> &amp;&amp; <name>month</name> &lt;= <name>DEC</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Winter semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>APR</name> &amp;&amp; <name>month</name> &lt;= <name>NOV</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Spring semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>NOV</name> &amp;&amp; <name>month</name> &lt;= <name>DEC</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Autumn semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>DEC</name> &amp;&amp; <name>month</name> &lt;= <name>MAR</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Winter semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>MAR</name> &amp;&amp; <name>month</name> &lt;= <name>APR</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Spring semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>APR</name> &amp;&amp; <name>month</name> &lt;= <name>NOV</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Autumn semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>NOV</name> &amp;&amp; <name>month</name> &lt;= <name>DEC</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Winter semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
<else>else if<condition>(<expr><name>month</name> &gt;= <name>DEC</name> &amp;&amp; <name>month</name> &lt;= <name>MAR</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Winter semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt></then>
&amp;&amp; <name>month</name> &lt;= <name>DEC</name>

<expr_stmt><expr><name>cout</name> &lt;&lt; "Fall semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

<else>

<expr_stmt><expr><name>cout</name> &lt;&lt; "Invalid month!" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

<comment type="block"></*
* Part 4: nested 'if'
*/
<expr_stmt><expr><name>cout</name> &lt;&lt; "\n------ part 4: ------" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

<comment type="block"></* YOUR CODE HERE */
<if><condition>(<expr><name>month</name> &lt; <name>MAY</name> ||
<name>month</name> &gt; <name>Aug</name>)</condition><then>

<expr_stmt><expr><name>cout</name> &lt;&lt; "Non-summer semester" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

<if><condition>(<expr><name>day</name> == <name>SUN</name> ||
<name>day</name> == <name>SAT</name>)</condition><then>

<expr_stmt><expr><name>cout</name> &lt;&lt; "It is a weekend" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

<else>

<expr_stmt><expr><name>cout</name> &lt;&lt; "It is a weekday" &lt;&lt; 
<name>endl</name></expr>;</expr_stmt>

</if>

</if>

<return>return <expr>0</expr>;</return>

</block></function>
</unit>
```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  ns="http://www.sdml.info/srcML/src" >
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional>
        <attribute name="language"/>
      </optional>
      <optional>
        <attribute name="filename"/>
      </optional>
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
      </oneOrMore>
      <optional>
        <text/>
      </optional>
      <ref name="using"/>
      <optional>
        <text/>
      </optional>
      <optional>
        <ref name="comment"/>
      </optional>
      <element name="function">
        <element name="type">
          <element name="name">
            <text/>
          </element>
        </element>
        <element name="name">
          <text/>
        </element>
        <element name="parameter_list">
          <text/>
        </element>
        <text/>
        <element name="block">
          <oneOrMore>
            <choice>
              <text/>
              <ref name="declarative_statement"/>
              <ref name="expression_statement"/>
              <ref name="comment"/>
              <ref name="if"/>
            </choice>
          </oneOrMore>
        </element>
      </element>
    </element>
  </start>
</grammar>
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.sdml.info/srcML/src" xmlns:cpp="http://www.sdml.info/srcML/cpp" language="C++" filename="complete-5.cpp">
  <comment type="block">/******************************************************************
** complete-5.cpp
* Purpose: Demonstrate more conditional statements
*******************************************************************/</comment>

  <cpp:include>
    #<cpp:directive>include</cpp:directive>
    <cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
  <using>using namespace <name>std</name>;</using>

  <function><type><name>int</name></type><name>main</name><parameter_list>()</parameter_list> <block>{
    <comment type="line">// Constant symbols to use throughout the program.</comment>
    <decl_stmt><decl><type><name>const</name> <name>int</name></type> <name>SUN</name> =<init> <expr>1</expr></init>, <name>MON</name> =<init> <expr>2</expr></init>, <name>TUE</name> =<init> <expr>3</expr></init>, <name>WED</name> =<init> <expr>4</expr></init>, <name>THU</name> =<init> <expr>5</expr></init>, <name>FRI</name> =<init> <expr>6</expr></init>, <name>SAT</name> =<init> <expr>7</expr></init></decl>;</decl_stmt>

    <decl_stmt><decl><type><name>const</name> <name>int</name></type> <name>JAN</name> =<init> <expr>1</expr></init>, <name>FEB</name> =<init> <expr>2</expr></init>, <name>MAR</name> =<init> <expr>3</expr></init>, <name>APR</name> =<init> <expr>4</expr></init>, <name>MAY</name> =<init> <expr>5</expr></init>, <name>JUN</name> =<init> <expr>6</expr></init>, <name>JUL</name> =<init> <expr>7</expr></init>, <name>AUG</name> =<init> <expr>8</expr></init>, <name>SEP</name> =<init> <expr>9</expr></init>, <name>OCT</name> =<init> <expr>10</expr></init>, <name>NOV</name> =<init> <expr>11</expr></init>, <name>DEC</name> =<init> <expr>12</expr></init></decl>;

    <comment type="line">// Variables to hold input.</comment>
    <decl_stmt><decl><type><name>int</name></type> <name>month</name>, <name>day</name>, <name>season</name></decl>;</decl_stmt>

    <decl_stmt><decl><type><name>bool</name></type> <name>is_morning_class</name> =<init> <expr>false</expr></init></decl>;</decl_stmt>

    <comment type="line">// Prompt user to enter month</comment>
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Type a number " &lt;&lt; <name>JAN</name> &lt;&lt; " to select a month:
    &lt;&lt; \\
    
    &lt;&lt; \\

    &lt;&lt; \\

    &lt;&lt; " to select a month:\n"</expr>;</expr_stmt>

    <expr_stmt><expr><name>cin</name> &gt;&gt; <name>month</name>;</expr_stmt>

    <if_stmt><cond><expr><name>is_morning_class</name></expr></cond>
      <block>{
        <comment type="line">// morning classes from January 4 to April 6
        <decl_stmt><decl><type><name>int</name></type> <name>MORNING_MONTHS</name> =<init> <expr>4</expr></init>, <name>APRIL</name> =<init> <expr>11</expr></init>, <name>SEPTEMBER</name> =<init> <expr>9</expr></init>, <name>LAHASA</name> =<init> <expr>12</expr></init></decl>;

        <comment type="line">// Prompt user to enter day</comment>
        <expr_stmt><expr><name>cout</name> &lt;&lt; "Enter day: 
        &lt;&lt; " to select a day:\n"</expr>;</expr_stmt>

        <expr_stmt><expr><name>cin</name> &gt;&gt; <name>day</name>;</expr_stmt>
      }</block>
    <else>
      <block>{
        <comment type="line">// Late class from May 7 to August 10
        <decl_stmt><decl><type><name>int</name></type> <name>ZONE</name> =<init> <expr>7</expr></init>, <name>DECEMBER</name> =<init> <expr>12</expr></init>, <name>SEPT</name> =<init> <expr>9</expr></init>, <name>INV</name> =<init> <expr>-1</expr></init></decl>;

        <comment type="line">// Prompt user to enter season</comment>
        <expr_stmt><expr><name>cout</name> &lt;&lt; "Enter season: 
        &lt;&lt; " to select a season:\n"</expr>;</expr_stmt>

        <expr_stmt><expr><name>cin</name> &gt;&gt; <name>season</name>;</expr_stmt>
      }</block>
    </if_stmt>
  </block>
</function>
</unit>
<name>month</name></expr></expr_stmt>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argu
ent><expr>20000</expr></argument>,
<argument><expr>'\n'</expr></argument>)</argument_list></call></expr_stmt>
<comment type="line">// prompt user to enter day of the week</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Type a number " &lt;&lt; <name>SUN</name> &lt;&lt; "-" &lt;&lt; <name>SAT</name>
 &lt;&lt; " to select a day (Sunday - Saturday)
&gt;"
</expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name> &gt;&gt; <name>day</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argu
ent><expr>20000</expr></argument>,
<argument><expr>'\n'</expr></argument>)</argument_list></call></expr_stmt>
<comment type="line">// prompt user to indicate if attending a morning class</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Are you attending a morning class?
(y/n)\n" &lt;&lt; "\n";
</expr>;</expr_stmt>
<decl_stmt><decl><type><name>char</name></type> <name>tempchar</name> =<init>
<expr><name>cin</name>.<call><name>get</name><argument_list>()</argument_list></call>
</expr></init></decl>;</decl_stmt>  
<comment type="line">// get a single character</comment>
<expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argu
ent><expr>20000</expr></argument>,
<argument><expr>'\n'</expr></argument>)</argument_list></call></expr_stmt>
<expr_stmt><expr><name>is_morning_class</name> = (<name>tempchar</name> == 'y' ||
<name>tempchar</name> == 'Y')</expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "You chose month: " &lt;&lt; <name>month</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "You chose day: " &lt;&lt; <name>day</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Is morning class: " &lt;&lt; <name>is_morning_class</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
<comment type="block">/*
 Part 1: chained else/if */</comment>
<if><condition>(<expr><name>day</name> == <name>MON</name> ||
<name>day</name> == <name>WED</name> || <name>day</name> ==
<name>FRI</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Schedule A" &lt;&lt;
</expr>;</expr_stmt></then>
<else>else <if><condition>(<expr><name>day</name> == <name>TUE</name> ||
<name>day</name> == <name>THU</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Schedule B" &lt;&lt;
</expr>;</expr_stmt></then>
<else>else <if><condition>(<expr><name>day</name> ==
<name>SAT</name>)</condition><then>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Schedule C" &lt;&lt;
</expr>;</expr_stmt></then>
<else>else

<expr_stmt><expr><name>cout</name> &lt;&lt; "No schedule" &lt;&lt; \n</expr>;</expr_stmt></if></else></if></if></if><br />

<comment type="block">/*
* Part 2: "switch" statement
*/</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; \n; "part 2: ------" &lt;&lt; \n</expr>;</expr_stmt>
<expr_stmt><name>cout</name> &lt;&lt; \n; "------ part 2: ------" &lt;&lt; \n</expr>;</expr_stmt>
<comment type="line">// leave this line here</comment>
<comment type="block">/* YOUR CODE HERE */</comment>
<switch>switch<condition>(<expr><name>month</name></expr>)</condition>
<br />
<case>case <expr><name>JAN</name></expr>:
</case><case>case <expr><name>FEB</name></expr>:
</case><case>case <expr><name>MAR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><expr_stmt><name>cout</name> &lt;&lt; "Spring semester\n"</expr>;</expr_stmt>
<br />
<expr_stmt><name>season</name> = <name>SPRING</name></expr>;</expr_stmt><break>break;</break><br />
<case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><expr_stmt><name>cout</name> &lt;&lt; "Summer semester\n"</expr>;</expr_stmt>
<br />
<expr_stmt><name>season</name> = <name>SUMMER</name></expr>;</expr_stmt><break>break;</break><br />
<case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><case>case <expr><name>APR</name></expr>:
</case><expr_stmt><name>cout</name> &lt;&lt; "Fall semester\n"</expr>;</expr_stmt>
<br />
<expr_stmt><name>season</name> = <name>FALL</name></expr>;</expr_stmt><break>break;</break><br />
<case>default:
</case><default>default: <expr_stmt><name>cout</name> &lt;&lt; \n; "Invalid month!\n"
</expr>;</expr_stmt>
<br />
<expr_stmt><name>season</name> = <name>INVALID</name></expr>;</expr_stmt><default>}</default>
<switch></switch><br />
<comment type="block">/*
* Part 3: ternary operator
*/</comment>
<expr_stmt><name>cout</name> &lt;&lt; \n; "Have a good "</expr>;</expr_stmt>
<expr_stmt><name>cout</name> &lt;&lt; (<name>is_morning_class</name> ? \"day\" : \"evening\")</expr_stmt>
```cpp
/* Part 4: nested conditions */

<<expr_stmt>><<expr><name>cout</name>&lt;&lt;&lt;<<name>endl</name>;</expr_stmt>
<<comment type="block">/#
* Part 4: nested conditions
*/</comment>

<<expr_stmt>><<expr><name>cout</name>&lt;&lt;"\n------ part 4: ------"&lt;&lt;&lt;<<name>endl</name>;</expr_stmt>   // leave this line here
<<comment type="block">/** YOUR CODE HERE **/</comment>

<<expr_stmt>><<expr><name>cout</name>&lt;&lt;"Parking will be "<expr_stmt>
<<if><condition>(<expr><name>month</name>&lt;&lt;&lt;&lt;<name>JAN</name> || <name>month</name>&gt;&lt;&lt;&lt;&lt;<name>DEC</name>)</condition><then>
<<expr_stmt><expr><name>cout</name>&lt;&lt;"imaginary\n";</expr_stmt></then>
<<else>else
<<block>{
<<if><condition>(<expr><name>season</name>&lt;&lt;&lt;&lt;SUMMER&name>)</condition><then>
<<if><condition>(<expr><name>day</name>&lt;&lt;&lt;&lt;SUN</name> || <name>day</name>&gt;&lt;&lt;&lt;&lt;SAT</name>)</condition><then>
<<expr_stmt><expr><name>cout</name>&lt;&lt;"imaginary";</expr_stmt></then>
<<else>else
<<expr_stmt><expr><name>cout</name>&lt;&lt;(<name>is_morning_class</name>?"moderate":"easy");</expr_stmt></else>
</if></if></if></block>
</else>

<<expr_stmt><expr><name>cout</name>&lt;&lt;"imaginary\n";</expr_stmt>
<<else>else
<<block>{
<<switch>switch<condition>(<expr><name>day</name>)</condition>
<<block>{
<<case>case <expr><name>SUN</name>:
<<break>break;</break>
<<case>case <expr><name>SAT</name>:
<<expr_stmt><expr><name>cout</name>&lt;&lt;"easy\n";</expr_stmt></case>
<<case>case <expr><name>MON</name>:
<<case>case <expr><name>TUE</name>:
<<case>case <expr><name>WED</name>:
<<case>case <expr><name>THU</name>:
<<expr_stmt><expr><name>cout</name>&lt;&lt;"hard\n";</expr_stmt></case>
<<default>default: <expr_stmt><expr><name>cout</name>&lt;&lt;"imaginary\n";</expr_stmt>
</default>}</block></switch>
</case></case></case></case></case>
</block></expr_stmt></expr_stmt>

/* Part 5: order of operations */

<<expr_stmt><expr><name>cout</name>&lt;&lt;"\n------ part 5: ------"&lt;&lt;&lt;<<name>endl</name>;</expr_stmt>   // leave this line here
```

```cpp
/** use parentheses here: **/ if (true || true) && false
{ cout << "AND is evaluated first" << endl; }
else { cout << "OR is evaluated first (test passed)" << endl; }

/** use parentheses here: **/ ! (true && false)
{ cout << "AND is evaluated first (test passed)" << endl; }
else { cout << "NOT is evaluated first" << endl; }

return 0;
```
Figure A.9 Lab 5 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
      </oneOrMore>
      <optional><text/></optional>
      <ref name="using"/>
      <optional><text/></optional>
      <optional><ref name="comment"/></optional>
      <element name="function">
        <element name="type">
          <element name="name">
            <text/>
          </element>
        </element>
        <text/>
        <element name="name">
          <text/>
        </element>
        <element name="parameter_list">
          <text/>
        </element>
        <text/>
        <element name="block">
          <oneOrMore>
            <choice>
              <text/>
              <ref name="declarative_statement"/>
              <ref name="expression_statement"/>
              <ref name="comment"/>
              <ref name="if"/>
              <ref name="switch"/>
            </choice>
          </oneOrMore>
        </element>
      </element>
    </element>
  </start>
</grammar>
```
<element name="return">
  <oneOrMore>
    <choice>
      <text/>
      <ref name="expression"/>
      <ref name="expression_statement"/>
    </choice>
  </oneOrMore>
</element>
<text/>
</element>
</optional>
</element>
</grammar>
Figure A.10 Lab 6 srcML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<comment type="block">/******************************************************************
**
* complete-6.cpp
* Purpose: Demonstrate conditional loop structures
*******************************************************************/

<cpp:include>
  #<cpp:directive>include</cpp:directive>
</cpp:include>
<cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<using>using namespace <name>std</name>;</using>

<comment type="line">// This is a function prototype. //</comment>
<comment type="line">// We will discuss them in later labs, but ignore it for now. //</comment>
<function_decl><type><name>int</name></type>
  <name>input_an_int</name><parameter_list>()</parameter_list>;</function_decl>

<function><type><name>int</name></type>
  <name>main</name><parameter_list>()</parameter_list> <block>{
    <comment type="line">// multipurpose reusable integer for loop counting. //</comment>
    <comment type="line">// remember to initialize before each use! //</comment>
    <decl_stmt><decl><type><name>int</name></type> <name>i</name></decl>;</decl_stmt>
    <comment type="line">// multipurpose reusable integer to hold user input. //</comment>
    <decl_stmt><decl><type><name>int</name></type>
      <name>num</name></decl>;</decl_stmt>

    /* Part 1: off-by-one */
    <expr_stmt><expr><name>cout</name> &lt;&lt; "------ part 1 (off by one): ------" &lt;&lt; endl</expr>;</expr_stmt>
    <expr_stmt><expr><name>i</name> = 0</expr>;</expr_stmt>
    <comment type="line">// this will loop from 0 to (n-1) //</comment>
    <while>while<condition>(<expr><name>i</name> &lt;= <name>num</name>)</condition> <block>{
      <expr_stmt><expr><name>cout</name> &lt;&lt; <name>i</name> &lt;&lt; " 
      </expr_stmt>  <expr_stmt><expr>++<name>i</name></expr>;</expr_stmt>
    }</block></while>

    /* Part 2: multiple part */
    <expr_stmt><expr><name>num</name> = <call><name>input_an_int</name><argument_list>()</argument_list></call></expr>;</expr_stmt>
    <comment type="line">// function call //</comment>
    <expr_stmt><expr><name>i</name> = 0</expr>;</expr_stmt>
    <comment type="line">// this will loop from 0 to (n-1) //</comment>
    <while>while<condition>({<expr><name>i</name>}</condition> <name>num</name>) &lt;=
      <block>{
        <expr_stmt><expr><name>cout</name> &lt;&lt; " 
        </expr_stmt>  <expr_stmt><expr>++<name>i</name></expr>;</expr_stmt>
      }</block></while>

    /* Part 3: check sum */
    <expr_stmt><expr><name>num</name> = <call><name>input_an_int</name><argument_list>()</argument_list></call></expr>;</expr_stmt>
    <comment type="line">// function call //</comment>
    <expr_stmt><expr><name>i</name> = 0</expr>;</expr_stmt>
    <comment type="line">// this will loop from 0 to (n-1) //</comment>
    <while>while<condition>({<expr><name>i</name>}</condition> <name>num</name>) &lt;=
      <block>{
        <expr_stmt><expr><name>cout</name> &lt;&lt; " 
        </expr_stmt>  <expr_stmt><expr>++<name>i</name></expr>;</expr_stmt>
        <expr_stmt><expr><name>cout</name> &lt;&lt; 
      </expr_stmt>  <expr_stmt><expr>++<name>i</name></expr>;</expr_stmt>
      }</block></while>
    <expr_stmt><expr><name>cout</name> &lt;&lt; endl</expr>;</expr_stmt>
  }</block></function>
</comment type="block">/#
```c
/* Part 2: accumulate a sum */
<expr_stmt><expr><name>cout</name> &lt;&lt; "\n------ part 2 (accumulator): ------"
&lt;&lt; <name>endl</name></expr>;</expr_stmt>

<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>num</name> =
<call><name>input_an_int</name></call></expr>;</expr_stmt>

<expr_stmt><expr><name>i</name> = <name>num</name></expr>;</expr_stmt>
<decl_stmt><decl><type><name>int</name></type> <name>sum</name> =</init>
<expr>0</expr></init></decl>;</decl_stmt>

<while>while<condition>(<expr><name>i</name> &gt; 0</expr>)</condition>
<block>{
<expr_stmt><expr><name>sum</name> += <name>i</name></expr>;</expr_stmt>
<expr_stmt><expr>--<name>i</name></expr>;</expr_stmt>
}</block></while>

<expr_stmt><expr><name>cout</name> &lt;&lt; "Sum of 0 to " &lt;&lt;
<name>num</name> &lt;&lt; ": " &lt;&lt; <name>sum</name> &lt;&lt;
<name>endl</name></expr>;</expr_stmt>

<comment type="block">/* Part 3: for loop */
<expr_stmt><expr><name>cout</name> &lt;&lt; "\n------ part 3 (sequence of squares): ------"
&lt;&lt; <name>endl</name></expr>;</expr_stmt>
<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>num</name> =
<call><name>input_an_int</name></call></expr>;</expr_stmt>

<for>for(<init><expr><name>i</name> = 0</expr>;</init>
<condition><expr><name>i</name> * <name>i</name> &lt;= <name>num</name></expr>;</condition> <incr><expr>++<name>i</name></expr></incr>)
<block>{
<expr_stmt><expr><name>cout</name> &lt;&lt; (<name>i</name> * <name>i</name>)
&lt;&lt; " \n ";</expr>;</expr_stmt>
}</block></for>

<expr_stmt><expr><name>cout</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>

<comment type="block">/* Part 4: do-while */
<expr_stmt><expr><name>cout</name> &lt;&lt; "\n------ part 4 (do-while): ------"
&lt;&lt; <name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "Enter a sequence of integers, then 0 when
finished."
&lt;&lt; <name>endl</name></expr>;</expr_stmt>
<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>sum</name> = 0</expr>;</expr_stmt>
<do>do
<block>{
<expr_stmt><expr><name>num</name> =
<call><name>input_an_int</name></call></expr>;</expr_stmt>
```

```cpp
<call><name>input_an_int</name><argument_list>()</argument_list></call>
<expr_stmt>
  <expr>
    <name>sum</name> += <name>num</name>
  </expr>
</expr_stmt>
</block>
while<condition>(<expr><name>num</name> != 0</expr>)</condition>;
<expr_stmt>
  <expr>
    <name>cout</name> &lt;&lt; "The sum is " &lt;&lt; <name>sum</name>
    &lt;&lt; <name>endl</name>
  </expr>
</expr_stmt> /* Part 5: nested loops */
<expr_stmt>
  <expr>
    <name>cout</name> &lt;&lt; "------ part 5 (nested chickens): ------"
    &lt;&lt; <name>endl</name>
  </expr>
</expr_stmt> /* YOUR CODE HERE */
<decl_stmt>
  <decl>
    <type><name>int</name></type> <name>hen</name>,
    <name>chick</name></decl>
</decl_stmt>
<decl_stmt>
  <decl>
    <type><name>int</name></type> <name>h</name>,
    <name>c</name></decl>
</decl_stmt>
<expr_stmt>
  <expr>
    <name>cout</name> &lt;&lt; "Enter the number of hens:"
    &lt;&lt; <name>endl</name>
  </expr>
</expr_stmt>
<expr_stmt>
  <expr>
    <name>cin</name> &gt;&gt; <name>hen</name>
  </expr>
</expr_stmt>
<expr_stmt>
  <expr>
    <name>cout</name> &lt;&lt; "Enter the number of chicks per hen:"
    &lt;&lt; <name>endl</name>
  </expr>
</expr_stmt>
<for>
  for (<init><expr><name>h</name> = 1</expr>;</init>
  <condition><expr><name>h</name> &lt;= <name>hen</name></expr>;</condition>
  <incr><expr><name>h</name>++</expr></incr>
  )
  <block>
    <expr_stmt>
      <expr>
        <name>cout</name> &lt;&lt; "CLUCK!"
      </expr>
    </expr_stmt>
    <expr_stmt>
      <expr>
        <name>endl</name>
      </expr>
    </expr_stmt>
    <for>
      for (<init><expr><name>c</name> = 1</expr>;</init>
      <condition><expr><name>c</name> &lt;= <name>chick</name></expr>;</condition>
      <incr><expr><name>c</name>++</expr></incr>
      )
      <block>
        <expr_stmt>
          <expr>
            <name>cout</name> &lt;&lt; "cheep!"
          </expr>
        </expr_stmt>
        <expr_stmt>
          <expr>
            <name>endl</name>
          </expr>
        </expr_stmt>
      </block>
    </for>
  </block>
</for>
<return>return <expr>0</expr>;</return>
</block>
```

// A function to get user input and try to interpret it as an integer. 
// Do not modify this.

```
<expr_stmt><expr><name>cout</name> &lt;&lt; "Type an integer, then press enter:\n\&gt;"
&lt;&lt; <name>flush</name></expr>:</expr_stmt>
  <expr_stmt><expr><name>cin</name> &gt;&gt; <name>num</name></expr>;</expr_stmt>
  <expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argumen
t><expr>20000</expr></argument>,
<argument><expr>\n</expr></argument>)</argument_list></call></expr>:</expr_stmt>
  <return>return <expr><name>num</name></expr>;</return>
</block></function>
</unit>
Figure A.11 Lab 6 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
         ns="http://www.sdml.info/srcML/src">  
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
      </oneOrMore>
      <optional><text/></optional>
      <ref name="using"/>
      <optional><text/></optional>
      <oneOrMore>
        <ref name="comment"/>
      </oneOrMore>
      <ref name="function_declaration"/>
      <optional><ref name="comment"/></optional>
      <element name="function">
        <element name="type">
          <element name="name">
            <text/>
          </element>
          <text/>
        </element>
        <element name="parameter_list">
          <text/>
        </element>
        <element name="block">
          <oneOrMore>
            <choice>
              <text/>
              <ref name="declarative_statement"/>
              <ref name="expression_statement"/>
              <ref name="comment"/>
              <ref name="if"/>
            </choice>
          </oneOrMore>
        </element>
      </element>
    </element>
  </start>
</grammar>
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
language="C++" filename="complete-7.cpp">
  <comment type="block">/******************************************************************
  **
  * complete-7.cpp
  * Purpose: Introduce concepts of functions
  *******************************************************************/</comment>
  <cpp:include>
    #<cpp:directive>include</cpp:directive>
    <cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
  <cpp:include>
    #<cpp:directive>include</cpp:directive>
    <cpp:file>&lt;string&gt;</cpp:file></cpp:include>
  </cpp:include>
  <using>using namespace <name>std</name>;</using>

  <comment type="line">// prototypes</comment>
  <function_decl>
    <type><name>int</name></type>
    <name>input_an_int</name><parameter_list>()</parameter_list>;</function_decl>
  <function_decl>
    <type><name>void</name></type>
    <name>print_sequence</name><parameter_list>(<param><decl><type><name>const</name>
        <name>int</name></type><name>upperLimit</name></decl></param>)</parameter_list>;</function_decl>
  <function_decl>
    <type><name>string</name></type>
    <name>rate_time</name><parameter_list>(<param><decl><type><name>const</name>
        <name>int</name></type><name>minutes</name></decl></param>)</parameter_list>;</function_decl>
  <function_decl>
    <type><name>void</name></type>
    <name>call_counter</name><parameter_list>()</parameter_list>;</function_decl>

  <function>
    <type><name>int</name></type>
    <name>main</name><parameter_list>()</parameter_list> <block>{
      <decl_stmt>
        <decl>
          <type><name>int</name></type> <name>n</name></decl>
      </decl_stmt>
      <comment type="line">// for user input</comment>
      <expr_stmt>
        <expr>
          <call><name>call_counter</name><argument_list>()</argument_list></call>
        </expr>
      </expr_stmt>
      <expr_stmt>
        <expr>
          <call><name>call_counter</name><argument_list>()</argument_list></call>
        </expr>
      </expr_stmt>
      <comment type="line">// first call</comment>
      <expr_stmt>
        <expr>
          <call><name>call_counter</name><argument_list>()</argument_list></call>
        </expr>
      </expr_stmt>
      <comment type="line">// using functions to get input and print output...</comment>
      <expr_stmt>
        <expr>
          <call><name>input_an_int</name><argument_list>()</argument_list></call>
        </expr>
      </expr_stmt>
      <expr_stmt>
        <expr>
          <call><name>print_sequence</name><argument_list>(<argument><name>n</name></argument>)</argument_list></call>
        </expr>
      </expr_stmt>
    } // end of main
  </function>
</unit>
```cpp
//second call

//Judging commute time using both functions.

//third call

//keep window open after output.

// Get user input and try to interpret it as an integer.

//Function to count from 0 to passed int.
```
&lt;for&gt;&lt;for_init&gt;&lt;decl&gt;&lt;type&gt;&lt;name&gt;int&lt;/name&gt;&lt;/type&gt;
&lt;name&gt;i&lt;/name&gt;&lt;equals&gt;&lt;init&gt;&lt;expr&gt;0&lt;/expr&gt;&lt;/init&gt;&lt;/decl&gt;&lt;/init&gt;&lt;condition&gt;&lt;expr&gt;&lt;name&gt;i&lt;/name&gt;&lt;less_than_or_equal_to&gt;&lt;init&gt;&lt;name&gt;upperLimit&lt;/name&gt;&lt;/init&gt;&lt;/condition&gt;&lt;increment&gt;&lt;expr&gt;1&lt;/expr&gt;&lt;/increment&gt;&lt;/block&gt;
{&lt;expr_stmt&gt;&lt;expr&gt;&lt;name&gt;cout&lt;/name&gt; &lt;lt;&lt; &lt;expr&gt;&lt;name&gt;i&lt;/name&gt; &lt;lt;&lt; &lt;expr&gt;" 
&"&lt;/expr&gt;&lt;/expr_stmt&gt;
}&lt;/block&gt;&lt;/for&gt;
&lt;expr_stmt&gt;&lt;expr&gt;&lt;name&gt;cout&lt;/name&gt; &lt;lt;&lt; &lt;name&gt;endl&lt;/name&gt;&lt;/expr&gt;&lt;/expr_stmt&gt;
}&lt;/block&gt;&lt;/function&gt;
&lt;function&gt;&lt;type&gt;&lt;name&gt;string&lt;/name&gt;&lt;/type&gt;
&lt;name&gt;rate_time&lt;/name&gt;&lt;parameter_list&gt;&lt;param&gt;&lt;decl&gt;&lt;type&gt;&lt;const&gt;&lt;name&gt;int&lt;/name&gt;&lt;/type&gt;&lt;name&gt;minutes&lt;/name&gt;&lt;/decl&gt;&lt;/param&gt;&lt;/parameter_list&gt;&lt;comment type="line"/&gt;&lt;/function&gt;
&lt;function&gt;&lt;type&gt;&lt;name&gt;void&lt;/name&gt;&lt;/type&gt;
&lt;name&gt;call_counter&lt;/name&gt;&lt;parameter_list&gt;&lt;/parameter_list&gt;&lt;comment type="line"/&gt;&lt;/function&gt;
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
<include href="../../src/includes/cpp-core.rng"/>
<start>
  <element name="unit">
    <optional><attribute name="language"/></optional>
    <optional><attribute name="filename"/></optional>
    <oneOrMore>
      <choice>
        <text/>
        <ref name="comment"/>
        <ref name="iostream"/>
        <ref name="iomanip"/>
        <ref name="cmath"/>
        <ref name="string"/>
      </choice>
    </oneOrMore>
    <ref name="using" />
    <ref name="comment"/>
    <oneOrMore>
      <text/>
      <ref name="function_declaration"/>
    </oneOrMore>
    <optional><ref name="comment"/></optional>
    <element name="function">
      <element name="type">
        <element name="name">
          <text/>
        </element>
      </element>
      <element name="name">
        <text/>
      </element>
      <element name="parameter_list">
        <text/>
      </element>
      <element name="block">
        <oneOrMore>
          <choice>
            <text/>
            <ref name="declarative_statement"/>
          </choice>
        </oneOrMore>
      </element>
    </element>
  </element>
</start>
<define>
  <element name="name">
    <text/>
  </element>
  <element name="parameter_list">
    <text/>
  </element>
  <element name="block">
    <text/>
    <ref name="declarative_statement"/>
    <ref name="expression_statement"/>
    <ref name="expression_statement"/>
    <ref name="expression_statement"/>
    <optional>
      <element name="return">
        <text/> <!-- return -->
        <ref name="expression"/>
        <text/> <!-- ; -->
      </element>
      <optional>
        <text/>
      </optional>
    </optional>
  </element>
</define>
Figure A.14 Lab 8 srcML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  language="C++" filename="complete-8.cpp">
  <comment type="block">
    /**************************************************************************
    **
    ** complete-8.cpp
    ** Purpose: Demonstrate advanced function parameter concepts
    **************************************************************************
  </comment>
  <cpp:include>
    <cpp:directive>include</cpp:directive>
    <cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
  <comment type="line">using namespace <name>std</name>;</comment>
  <comment type="line"></comment>

  <function_decl><type><name>int</name></type><name>input_int</name><parameter_list>();</function_decl>
  <function_decl><type><name>double</name></type><name>input_double</name><parameter_list>();</function_decl>
  <function_decl><type><name>void</name></type><name>input_int</name><parameter_list>(<param><decl><type><name>int</name>&amp;</type></decl>,</param><param><decl><type><name>int</name>&amp;</type></decl>,</param><param><decl><type><name>int</name>&amp;</type></decl>)</parameter_list>;</function_decl>
  <function_decl><type><name>void</name></type><name>print_sequence</name><parameter_list>(<param><decl><type><name>int</name></type></decl>,</param><param><decl><type><name>int</name></type></decl>,</param><param><decl><type><name>int</name></type><name>step</name></decl>)</parameter_list>;</function_decl>
  <function_decl><type><name>void</name></type><name>print_sequence</name><parameter_list>(<param><decl><type><name>double</name></type></decl>,</param><param><decl><type><name>double</name></type></decl>)</parameter_list>;</function_decl>

  <function><type><name>int</name></type><name>main</name><parameter_list>();</function>

  {<decl_stmt>
    <decl><type><name>int</name></type><name>m</name></decl>,</decl_stmt>
  <decl_stmt>
    <decl><type><name>int</name></type><name>n</name></decl></decl_stmt>
  <decl_stmt>
    <decl><type><name>double</name></type><name>dm</name></decl></decl_stmt>
  <decl_stmt>
    <decl><type><name>int</name></type><name>step</name></decl></decl_stmt>

  /* Output sequence from startval to endval, entered by user */
  <expr_stmt><expr><call><name>input_int</name><argument_list>(<argument><expr><name>m</name></expr></argument>,<argument><expr><name>n</name></expr></argument>,</argument_list></call></expr>
  <expr_stmt><expr><call><name>input_double</name><argument_list>(<argument><expr><name>dm</name></expr></argument>,<argument><expr><name>dn</name></expr></argument>)</argument_list></call></expr>
  <expr_stmt><expr><call><name>print_sequence</name><argument_list>(<argument><expr><name>m</name></expr></argument>,<argument><expr><name>n</name></expr></argument>,<argument><expr><name>step</name></expr></argument>)</argument_list></call></expr>
  <expr_stmt><expr><call><name>print_sequence</name><argument_list>(<argument><expr><name>dm</name></expr></argument>,<argument><expr><name>dn</name></expr></argument>)</argument_list></call></expr>
</function>
</unit>
```
// move the following lines into a new function:

<call><name>print_sequence</name><argument_list>(<argument><expr><name>m</name></expr>,<argument><expr><name>n</name></expr>,<argument><expr><name>step</name></expr></argument_list></call></expr>;

<expr_stmt><expr><name>cout</name> &lt;&lt; "Decimal sequence from BEGIN to END
";</expr_stmt>

<expr_stmt><expr><name>dm</name> =<call><name>input_double</name><argument_list>()</call></expr>;</expr_stmt>

<expr_stmt><expr><name>dn</name> =<call><name>input_double</name><argument_list>()</call></expr>;</expr_stmt>

<expr_stmt><expr><call><name>print_sequence</name><argument_list>(<argument><expr><name>dm</name></expr>,<argument><expr><name>dn</name></expr></argument_list></call></expr>;

<return>return <expr>0</expr>;</return>

// Get user input and try to interpret it as an integer.

<function><type><name>int</name></type><name>input_int</name><parameter_list>()</parameter_list><block>{{
  <decl_stmt><decl><type><name>int</name></type><name>num</name> =<init><expr>0</expr></init></decl>;
  <expr_stmt><expr><name>cin</name> &gt;&gt; <name>num</name></expr>;
  <expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argument><expr>20000</expr></argument>,<argument><expr>'\n'</expr></argument>)</call></expr>;
  <return>return <expr><name>num</name></expr>;</return>
}}

//print_sequence

<function><type><name>void</name></type><name>print_sequence</name><parameter_list>(<param><decl><type><name>int</name></type><name>m</name></decl></param>, <param><decl><type><name>int</name></type><name>n</name></decl></param>, <param><decl><type><name>int</name></type><name>step</name></decl></param>)</parameter_list><call></call></function>
```cpp
<name>n</name>, <param><decl><type><name>int</name></type><name>step</name></param>)</parameter_list><block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Int sequence from " &lt;&lt; <name>m</name> &lt;&lt; " to " &lt;&lt; <name>n</name> &lt;&lt; ": " &lt;&lt; <name>endl</name></expr>;</expr_stmt>
    <for><init><decl><type><name>int</name></type><name>i</name>=<init><expr><name>m</name></expr></init></decl>;</init><condition><expr><name>i</name>&lt;=<name>n</name></expr>;</condition><incr><expr><name>i</name> += <name>step</name></expr></incr>)<block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; <name>i</name> &lt;&lt; "</expr>;</expr_stmt>
    <expr_stmt><expr><name>endl</name></expr>;</expr_stmt>
}</block></for>
} //print seq double
</function>
<function><type><name>void</name></type><name>print_sequence</name><parameter_list>(<param><decl><type><name>double</name></type><name>dm</name></decl></param>, <param><decl><type><name>double</name></type><name>dn</name></decl></param>)</parameter_list><block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Dec sequence from " &lt;&lt; <name>dm</name> &lt;&lt; " to " &lt;&lt; <name>dn</name> &lt;&lt; ": " &lt;&lt; <name>endl</name></expr>;</expr_stmt>
    <for><init><decl><type><name>double</name></type><name>i</name>=<init><expr><name>dm</name></expr></init></decl>;</init><condition><expr><name>i</name>&lt;=<name>dn</name></expr>;</condition><incr><expr><name>i</name> += 1</expr></incr>)<block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; <name>i</name> &lt;&lt; "</expr>;</expr_stmt>
    <expr_stmt><expr><name>endl</name></expr>;</expr_stmt>
}</block></for>
} //passing argument
</function>
<function><type><name>void</name></type><name>input_int</name><parameter_list>(<param><decl><type><name>int</name></type>&<name>num1</name></decl></param>, <param><decl><type><name>int</name></type>&<name>num2</name></decl></param>, <param><decl><type><name>int</name></type>&<name>num3</name></decl></param>)</parameter_list> <block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Type in 3 integers, then press enter:\n\&gt;"</expr>;</expr_stmt>
    <expr_stmt><expr><name>cin</name> &gt;&gt; <name>num1</name> &gt;&gt; <name>num2</name> &gt;&gt; <name>num3</name></expr>;</expr_stmt>
    <expr_stmt><expr><name>cin</name>.<call><name>ignore</name><argument_list>(<argumen
```
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  ns="http://www.sdml.info/srcML/src" >
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>
      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
      </oneOrMore>
      <ref name="using"/>
      <ref name="comment"/>
      <oneOrMore>
        <text/>
        <ref name="function_declaration"/>
      </oneOrMore>
      <optional><ref name="comment"/></optional>
    </element>
    <element name="function">
      <element name="type">
        <element name="name">
          <text/>
        </element>
        <text/>
        <element name="name">
          <text/>
        </element>
      </element>
      <text/>
      <element name="parameter_list">
        <text/>
      </element>
      <element name="block">
        <oneOrMore>
          <choice>
            <text/>
            <ref name="declarative_statement"/>
            <ref name="expression_statement"/>
            <ref name="comment"/>
            <ref name="if"/>
            <ref name="switch"/>
          </choice>
        </oneOrMore>
      </element>
    </element>
  </start>
</grammar>
<ref name="while"/>
<ref name="for"/>
<ref name="do"/>
</choice>
</oneOrMore>
<optional>
<element name="return">
<oneOrMore>
<choice>
<text/>
<ref name="expression"/>
<ref name="expression_statement"/>
</choice>
</oneOrMore>
</element>
</optional>
<text/>
</element>
<!-- end main function -->
<text/>
<!-- Extra functions -->
<oneOrMore>
<choice>
<text/>
<ref name="comment"/>
<ref name="function"/>
</choice>
</oneOrMore>
</element>
</start>
<define name="extra_function">
<zeroOrMore>
<ref name="comment"/>
</zeroOrMore>
<element name="function">
<element name="type">
<element name="name">
<text/>
</element>
</element>
<text/>
<element name="name">
<text/>
</element>
<element name="parameter_list">
<text/>
</element>
<element name="parameter_list">
<text/>
</element>
</element>
</define>
Figure A.16 Lab 9 RelaxNG

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
language="C++" filename="complete-9.cpp"><comment
type="block">******************************************************************************
* complete-9.cpp
* Purpose: Introduce the creation and use of arrays
******************************************************************************</comment>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<using>using namespace <name>std</name>;</using>
<comment type="line">//Prototype</comment>
<function_decl><type><name>void</name></type><name>print_array</name><parameter_list>(<param><decl><type><name>double</name>[]</t
ype></decl></param>,<param><decl><type><name>int</name></type></decl></param>)</parameter_list>;</function_decl>
<comment type="line">// Variables.</comment>
<decl_stmt><decl><type><name>double</name></type><name>total</name>=<init><expr>0</expr></init></decl>;</decl_stmt>
<function><type><name>int</name></type><name>main</name><parameter_list>()</parameter_list> <block>{
<decl_stmt><decl><type><name>const</name> <name>int</name></type><name>ARRAYLEN</name> =<init> <expr>10</expr></init></decl>;</decl_stmt>
<comment type="block">/* Do part 1 here: */</comment>
<decl_stmt><decl><type><name>double</name></type><name>numbers</name><index>[<expr>ARRAYLEN</expr>]</index>=<init> <expr><block>{<expr>0.0</expr>, <expr>1.1</expr>, <expr>2.2</expr>,<expr>3.3</expr>, <expr>4.4</expr>, <expr>5.5</expr>, <expr>6.6</expr>, <expr>7.7</expr>,
<expr>8.8</expr>, <expr>9.9</expr>}</block></expr></init></decl>;</decl_stmt>
<comment type="block">/* Do part 2 here: */</comment>
<expr_stmt><expr><name>cout</name>&lt;&lt;"The last element's value is:" &lt;&lt;<name>numbers</name><index>[<expr>9</expr>]</index>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<comment type="block">/* Do parts 3 and 4 here: */</comment>
<expr_stmt><expr><call><name>print_array</name><argument_list>(<argument><expr><name>numbers</name></expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<comment type="block">/* Do part 5 here: */</comment>
<for><init><decl><type><name>int</name></type><name>i</name>=<init><expr>0</expr></init></decl>;</init>
<condition><expr><name>i</name>&lt; <name>ARRAYLEN</name></expr>;</condition>
<block>
<decl_stmt><decl><type><name>int</name></type><name>s</name>=<init><expr>0</expr></init></decl>;
<for><init><decl><type><name>int</name></type><name>num</name>=<init><expr>numbers</expr></init></decl>;</init>
<condition><expr><name>num</name>&lt; <name>ARRAYLEN</name></expr>;</condition>
<block>
<decl_stmt><decl><type><name>int</name></type><name>flag</name>=<init><expr>0</expr></init></decl>;
<comment type="block">/* Do part 6 here: */</comment>
<call><name>print_array</name><argument_list>(<argument><expr><name>numbers</expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call>;
<call><name>print_array</name><argument_list>(<argument><expr><name>numbers</expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call>;
<condition><expr><name>i</name>&lt; <name>ARRAYLEN</name></expr>;</condition>
</for>
</for>
</block>
</expr_stmt>
</function>
</unit>
```
for (int i = 0; i < ARRAYLEN; i++) {
    numbers[index] = (10 * numbers[index]) +
}

for (int j = 0; j < ARRAYLEN; j++) {
    total += numbers[index]
}

cout << "Sum of array elements:" << total << endl;

print_array(input, length);

return 0;

void print_array(double input[], int length) {
    for (int i = 0; i < length; i++) {
        cout << input[i] << " ";
    }
    cout << endl;
}

char who[] = "Charles Babbage";

cout << who << endl;
Figure A.17 Lab 9 RelaxNG

```xml
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
     xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
     ns="http://www.sdml.info/srcML/src" >
   <include href="../../src/includes/cpp-core.rng"/>
   <start>
     <element name="unit">
       <optional><attribute name="language"/></optional>
       <optional><attribute name="filename"/></optional>
       <oneOrMore>
         <choice>
           <text/>
           <ref name="comment"/>
           <ref name="iostream"/>
           <ref name="iomanip"/>
           <ref name="cmath"/>
           <ref name="string"/>
         </choice>
       </oneOrMore>
       <ref name="using"/>
       <ref name="comment"/>
       <ref name="function_declaration"/>
       <!-- global variable declaration -->
       <ref name="comment"/>
       <ref name="declarative_statement"/>
       <optional><ref name="comment"/></optional>
     <element name="function">
       <element name="type">
         <element name="name">
           <text/>
         </element>
       </element>
       <text/>
       <element name="name">
         <text/>
       </element>
       <element name="parameter_list">
         <text/>
       </element>
       <element name="block">
         <oneOrMore>
           <choice>
             <text/>
             <ref name="declarative_statement"/>
             <ref name="expression_statement"/>
             <ref name="comment"/>
             <ref name="if"/>
             <ref name="switch"/>
           </choice>
         </oneOrMore>
       </element>
     </element>
   </start>
</grammar>
```
while
for
do

<optional>
<element name="return">
<oneOrMore>
<choice>
<text/>
<ref name="expression"/>
<ref name="expression_statement"/>
</choice>
</oneOrMore>
</element>
</optional>
<text/>

<!-- end main function -->
<text/>
<!-- Extra functions -->
<oneOrMore>
<choice>
<text/>
<ref name="comment"/>
<ref name="function"/>
</choice>
</oneOrMore>
</start>
<define name="extra_function">
<zeroOrMore>
<ref name="comment"/>
</zeroOrMore>
<element name="function">
<element name="type">
<element name="name">
<text/>
</element>
</element>
<text/>
<element name="name">
<text/>
</element>
<element name="parameter_list">
<text/>
</element>
</define>
<element name="block">
  <text/>
  <ref name="declarative_statement"/>
  <ref name="expression_statement"/>
  <ref name="expression_statement"/>
  <ref name="expression_statement"/>
  <optional>
    <element name="return">
      <text> <!-- return --></text>
      <ref name="expression"/>
      <text> <!-- ; --></text>
    </element>
  </optional>
  <text/>
</element>
</define>
</element>
</grammar>
Figure A.18 Lab 10 srcML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
<unit xmlns="http://www.sdml.info/srcML/src" xmlns:cpp="http://www.sdml.info/srcML/cpp" language="C++" filename="010/students/uanet/Labs/Lab10/MoreArrays.cpp"> <comment type="block">******************************************************************
** * Lab 10: MoreArrays.cpp
* Purpose: Demonstrate array concepts
******************************************************************</comment>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;string&gt;</cpp:file></cpp:include>
<cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>&lt;iomanip&gt;</cpp:file></cpp:include>
<using>using namespace <name>std</name>;</using>
<comment type="line">// Prototype</comment>
<function_decl><type><name>void</name></type>
<name>print_array</name><parameter_list>(<param><decl><type><name>int</name>[]</type></decl></param>,
<param><decl><type><name>int</name></type></decl></param>)</parameter_list>;</function_decl>
<function_decl><type><name>void</name></type>
<name>input_array</name><parameter_list>(<param><decl><type><name>int</name>[]</type></decl></param>,
<param><decl><type><name>int</name></type></decl></param>)</parameter_list>;</function_decl>
<function_decl><type><name>bool</name></type>
<name>arrays_equal</name><parameter_list>(<param><decl><type><name>int</name>[]</type></decl></param>,
<param><decl><type><name>int</name>[]</type></decl></param>,
<param><decl><type><name>int</name></type></decl></param>)</parameter_list>;</function_decl>
<comment type="block">/* implement for part 0 */</comment>
<function><type><name>void</name></type>
<name>print_array</name><parameter_list>(<param><decl><type><name>int</name>[]</type><name><name>arr</name><index>[]</index></name></decl></param>,
<param><decl><type><name>int</name></type><name>len</name></decl></param>)</parameter_list> <block>{
<for><init><decl><type><name>int</name></type><name>i</name></decl></init> <condition><expr><name>i</name>&lt;</expr><name>len</name></condition> <incr><expr><name>i</name>++</expr></incr>)
<block>{
<expr_stmt><expr><name>cout</name>&lt;&lt;<name><name>arr</name><index>[<expr><name>i</name></expr>]</name>&lt;&lt;<name>" 
"</name>&lt;&lt;</expr_stmt>
<block>}</n
```
```cpp
/* implement for part 1 */

#include <iostream>

void input_array(int input[], int ARRAYLEN) {
    std::cout << "Please enter the value for the array.\n";
    for (int i = 0; i < ARRAYLEN; ++i) {
        std::cin >> input[i];
    }
}

/* implement for part 2 */

bool arrays_equal(int numbers[], int inputs[], int ARRAYLEN) {
    bool equal = true;
    for (int i = 0; i < ARRAYLEN; ++i) {
        if (numbers[i] == inputs[i]) {
            equal = true;
        } else {
            equal = false;
        }
    }
    return equal;
}

/* implement for part 3 */

void print_table(string people[], int inputs[], int ARRAYLEN) {
    if (arrays_equal(inputs, people, ARRAYLEN)) {
        std::cout << "Equal\n";
    } else {
        std::cout << "Not equal\n";
    }
}
```
for (int i = 0; i < ARRAYLEN; ++i) {
    cout << left << setw(10) << people[i] << right << setw(10) << inputs[i] << endl;
}

void print_array(int ar[], int h, int w) {
    for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; ++j) {
            cout << ar[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    const int ARRAYLEN = 5;
    int numbers[ARRAYLEN] = {1, 4, 9, 16, 25};
    int inputs[ARRAYLEN] =;
    // ...
```cpp
<decl_stmt><decl><type><name>string</name></type><name><name>people</name><index>[]</index></name> =<init> <expr><block>{<expr>"Big Bird"</expr>,<expr>"Bert"</expr>,<expr>"Ernie"</expr>,<expr>"Grover"</expr>,<expr>"Oscar"</expr>}</block></init></decl>;</decl_stmt>
<expr_stmt><expr><call><name>print_array</name><argument_list>(<argument><expr><name>people</name></expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call></expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "---- Part 1: passing arrays ----
\n"</expr>;</expr_stmt><if><condition><expr><call><name>arrays_equal</name><argument_list>(<argument><expr><name>people</name></expr></argument>,<argument><expr><name>inputs</name></expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call></expr></condition><then> <block>{
  <expr_stmt><expr><name>cout</name> &lt;&lt; "Arrays are equal" &lt;&lt;
\n"</expr>;</expr_stmt>
}</block></then>
<else>else <block>{
  <expr_stmt><expr><name>cout</name> &lt;&lt; "Arrays are not equal" &lt;&lt;
\n"</expr>;</expr_stmt>
}</block></else></if>
<expr_stmt><expr><call><name>print_table</name><argument_list>(<argument><expr><name>people</name></expr></argument>,<argument><expr><name>inputs</name></expr></argument>,<argument><expr><name>ARRAYLEN</name></expr></argument>)</argument_list></call></expr_stmt>
<expr_stmt><expr><call><name>print_array</name><argument_list>(<argument><expr><name>matrix</name></expr></argument>, <argument><expr>3</expr></argument>,<argument><expr>4</expr></argument>)</argument_list></call></expr_stmt>
```

125
Figure A.19 Lab 10 RelaxNG

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
  <include href="../../src/includes/cpp-core.rng"/>
  <start>
    <element name="unit">
      <optional><attribute name="language"/></optional>
      <optional><attribute name="filename"/></optional>

      <oneOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="iostream"/>
          <ref name="iomanip"/>
          <ref name="cmath"/>
          <ref name="string"/>
        </choice>
      </oneOrMore>
      <optional><text/></optional>
      <ref name="using"/>
      <optional><text/></optional>
      <zeroOrMore>
        <choice>
          <text/>
          <ref name="comment"/>
          <ref name="function_declaration"/>
        </choice>
      </zeroOrMore>
    </element>
  </start>
</grammar>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
language="C++" filename="complete-11.cpp"><comment type="block">/*
complete-11.cpp
Binary Search and Selection Sort
*/</comment>
<cpp:include><cpp:directive>include</cpp:directive></cpp:include><cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<using>using namespace <name>std</name>;</using>
<comment type="line">// constant value to use throughout the program</comment>
<decl_stmt><decl><type><name>const</name> <name>int</name></type><name>SIZEOFARRAYS</name> =<init> <expr>10</expr></init></decl>;</decl_stmt>
<comment type="block">/*
Binary Search
returns the integer value of the location where the value was
found.
if not found, the function returns -1
*/</comment>
<function><type><name>int</name></type><name>binarySearch</name><parameter_list>(<param><decl><type><name>int</name></type><name>ar</name><index>[]</index></decl></param>,
<param><decl><type><name>int</name></type><name>valueSearchedFor</name></decl></param>)</parameter_list> <block>{
<decl_stmt><decl><type><name>int</name></type><name>firstIndex</name> =<init> <expr>0</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>int</name></type><name>lastIndex</name> =<init> <expr>(<name>SIZEOFARRAYS</name> -1)</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>bool</name></type><name>found</name> =<init> <expr>false</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>int</name></type><name>position</name> =<init> <expr>-1</expr></init></decl>;</decl_stmt>
<while>while <condition>(<expr><name>found</name> != true &amp;&amp; <name>firstIndex</name> &lt;= <name>lastIndex</name>)</expr>) <block>{
<decl_stmt><decl><type><name>int</name></type><name>middle</name> =<init> <expr>((<name>firstIndex</name> + <name>lastIndex</name>) / 2)</expr></init></decl_stmt>
<if>if <condition>(<expr><name>ar</name><index>[<expr><name>middle</name></expr>]</expr> == <name>valueSearchedFor</name>)</condition><then> <block>{
<expr_stmt><expr><name>position</name> = <name>middle</name></expr_stmt>
<expr_stmt><expr><name>found</name> = true</expr_stmt>
</block></then>
<else>else <if>if (</expr><name>firstIndex</name> &lt;= <name>lastIndex</name>)} / 2</expr>)<expr_stmt><expr><name>position</name> = <init> <expr>-1</expr></init></expr_stmt>
<while>while <condition>(<expr><name>found</name> != true &amp;&amp; <name>firstIndex</name> &lt;= <name>lastIndex</name>)</expr>) <block>{
<decl_stmt><decl><type><name>int</name></type><name>middle</name> =<init> <expr>((<name>firstIndex</name> + <name>lastIndex</name>) / 2)</expr></init></decl_stmt>
<if>if <condition>(<expr><name>ar</name><index>[<expr><name>middle</name></expr>]</expr> == <name>valueSearchedFor</name>)</condition><then> <block>{
<expr_stmt><expr><name>position</name> = <name>middle</name></expr_stmt>
<expr_stmt><expr><name>found</name> = true</expr_stmt>
</block></then>
<else>else <if>if
Selection Sort

given the array, sort in place using pseudocode provided in the instructions

/*
Selection Sort
*/

<function><type><name>void</name></type><name>selectionSort</name><parameter_list>(<param><decl><type><name>int</name></type><name>values</name><index>[]</index></name></decl></param>)<block>{
<for>for (<init><decl><type><name>int</name></type><name>startScan</name> =<init><expr>0</expr></init></decl>;</init><condition><expr><name>startScan</name> &lt; (NAME_SIZEOFARRAYS -1)</expr></condition><incr><expr><name>startScan</name>++</expr></incr>) <block>{
<decl_stmt><decl><type><name>int</name></type><name>index</name> =<init><expr><name>startScan</name></expr></init></decl>
<decl_stmt><decl><type><name>int</name></type><name>maxIndex</name> =<init><expr><name>startScan</name></expr></init></decl>
<decl_stmt><decl><type><name>int</name></type><name>maxValue</name> =<init><expr><name>values</name><index>[<expr><name>startScan</name></expr>]</expr></init></decl>
<for>for (<init><decl><type><name>int</name></type><name>index</name> =<init><expr>(<name>startScan</name> +1)</expr></init></decl>;</init><condition><expr><name>index</name> &lt; NAME_SIZEOFARRAYS</expr></condition><incr><expr><name>index</name>++</expr></incr>)<block>{
<if>if (<condition>(<expr><name>values</name><index>[<expr><name>index</name></expr>]</expr> &gt; <name>maxValue</name>)</condition><then> <block>{
<expr_stmt><expr><name>maxValue</name> =<expr><name>values</name><index>[<expr><name>index</name></expr>]</expr></expr_stmt>
<expr_stmt><expr><name>maxIndex</name> =<expr><name>index</name></expr></expr_stmt>
</block></then></if>
</block></for>
<expr_stmt><expr><name><name>values</name><index>[<expr><name>maxIndex</name></expr>]</index></name> =<expr><name>values</name><index>[<expr><name>startScan</name></expr>]</expr></expr_stmt>
</block></for></block>
<return><expr><name>position</name></expr>;</return>
</block></function>
values[index] = maxValue;
}
}

// Do Not Change the main function

int main() {

values[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

largeValues[] = {2200, 356, 12121, 44, 73, 8300, 31812, 1, 42, 3};

squareValues[] = {36, 100, 16, 81, 121, 9, 4, 64, 1, 25};

cout << "------     Section 1     -----
";
// testing Binary Search
int result = binarySearch(values, 20);
if (result == 1) {
    cout << "Section 1: Test Case 1 of 3 - pass\n";
} else {
    cout << "Section 1: Test Case 1 of 3 - fail\n";
}
result = binarySearch(values, 12);
if (result != -1) {
    cout << "Section 1: Test Case 2 of 3 - fail\n";
} else {
    cout << "Section 1: Test Case 2 of 3 - pass\n";
}
result = binarySearch(values, 2);
if (result == 1) {
    cout << "Section 1: Test Case 3 of 3 - pass\n";
} else {
    cout << "Section 1: Test Case 3 of 3 - fail\n";
}

return 0;
}
<call><name>binarySearch</name><argument_list>(<argument><expr><name>values</name></expr>,</argument><argument><expr>100</expr></argument>)</argument_list></call>
<if><condition><expr><name>result</name> == 9</expr></condition><then><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 1: Test Case 3 of 3 - fail\n"</expr_stmt>
</block></then><else><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 1: Test Case 3 of 3 - pass\n"
</block></else></if>
<expr_stmt><expr><call><name>selectionSort</name><argument_list>(<argument><expr><name>values</name></expr>)</argument_list></call></expr>
<expr_stmt><expr><call><name>selectionSort</name><argument_list>(<argument><expr><name>largeValues</name></expr>)</argument_list></call></expr>
<expr_stmt><expr><call><name>selectionSort</name><argument_list>(<argument><expr><name>squareValues</name></expr>)</argument_list></call></expr>
<if><condition><expr><name>values[0]</name> == 100</expr></condition><then><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 1 of 6 - pass\n"
</block></then><else><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 1 of 6 - fail\n"
</block></else></if>
<if><condition><expr><name>largeValues[0]</name> == 31812</expr></condition><then><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 3 of 6 - pass\n"
</block></then><else><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 3 of 6 - fail\n"
</block></else></if>
<if><condition><expr><name>largeValues[9]</name> == 1</expr></condition><then><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - pass\n"
</block></then><else><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - fail\n"
</block></else></if>
<if><condition><expr><name>largeValues[9]</name> == 1</expr></condition><then><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - pass\n"
</block></then><else><block>{
  <expr_stmt><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - fail\n"
</block></else></if>
```cpp
<if>if <condition>(<expr><name><name>squareValues</name><index>[<expr>0</expr>]</index></name> == 121</expr>)</condition><then> <block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - pass\n"</expr>;</expr_stmt>
}</block></then><else>else <block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Section 2: Test Case 4 of 6 - fail\n"</expr>;</expr_stmt>
}</block></else></if>

<if>if <condition>(<expr><name><name>squareValues</name><index>[<expr>9</expr>]</index></name> == 1</expr>)</condition><then> <block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Section 2: Test Case 6 of 6 - pass\n"</expr>;</expr_stmt>
}</block></then><else>else <block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; "Section 2: Test Case 6 of 6 - fail\n"</expr>;</expr_stmt>
}</block></else></if>

<return>return 0;</return>
</block></function>
```
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
    <include href="../../src/includes/cpp-core.rng"/>
    <start>
        <element name="unit">
            <optional><attribute name="language"/></optional>
            <optional><attribute name="filename"/></optional>
            <oneOrMore>
                <choice>
                    <text/>
                    <ref name="comment"/>
                    <ref name="iostream"/>
                    <ref name="iomanip"/>
                    <ref name="cmath"/>
                    <ref name="string"/>
                </choice>
            </oneOrMore>
        </element>
        <optional><text/></optional>
        <ref name="using"/>
        <optional><text/></optional>
        <!-- Constant to be used -->
        <ref name="comment"/>
        <ref name="declarative_statement"/>
        <!-- Binary search -->
        <ref name="comment"/>
        <ref name="function"/>
        <!-- Selection sort -->
        <ref name="comment"/>
        <ref name="function"/>
        <!-- Pre-populated testing -->
        <optional><ref name="comment"/></optional>
        <element name="function">
            <element name="type">
                <element name="name">
                    <text/>
                </element>
            </element>
            <element name="name">
                <text/>
            </element>
        </element>
        <text/>
        <element name="name">
            <text/>
        </element>
        <element name="parameter_list">
            <text/>
        </element>
        <text/>
        <element name="block">
            <oneOrMore>
        </grammar>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
language="C++" filename="complete-12.cpp"><comment
type="block">******************************************************************
**
* complete-12.cpp
* Purpose: Demonstrate the use of pointers
******************************************************************

<comment type="line"></comment>
<cpp:file>&lt;iostream&gt;</cpp:file></cpp:include>
<using>using namespace std;</using>
<comment type="line">// function prototypes</comment>
<function_decl><type><name>void</name></type>
<name>print_array</name><parameter_list>(<param><decl><type><name>int</name>
*</type></decl></param>,
<param><decl><type><name>int</name></type></decl></param>)</parameter_list>;</function_decl>
<function_decl><type><name>void</name></type>
<name>print_address_and_value</name><parameter_list>(<param><decl><type><name>const
</name> <name>int</name> *</type></decl></param>)</parameter_list>;</function_decl>
<function><type><name>int</name></type>
<name>main</name><parameter_list>()</parameter_list> <block>{
<decl_stmt><decl><type><name>int</name></type> <name>x</name> =<init>
<expr>2</expr></init></decl>;
<decl_stmt><decl><type><name>int</name>
*</type><name>ptr</name></decl>;</decl_stmt> <comment type="line">// declaring pointer to int</comment>
<decl_stmt><decl><type><name>const
</name> <name>int</name></type>
<name>SEQUENCE_LENGTH</name> =<init> <expr>8</expr></init></decl>;</decl_stmt>
<decl_stmt><decl><type><name>int</name></type>
<name><name>seq</name><index>[<expr><name>SEQUENCE_LENGTH</name></expr>]</index></name> =<init>
<expr><block>{<expr>10</expr>,<expr>20</expr>,<expr>30</expr>,<expr>40</expr>,<expr>
50</expr>,<expr>60</expr>,<expr>70</expr>,<expr>80</expr>}</block></expr></init></decl
>;</decl_stmt>
<comment type="block">/**** Part 1: get an address ****/</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "---- Part 1: get an address ----
"</expr_stmt>
<expr_stmt><expr><name>ptr</name> = &amp;<name>x</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "ptr now holds the address: " &lt;&lt;
<name>ptr</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
<comment type="block">/**** Part 1: get an address ****/</comment>
<expr_stmt><expr><name>cout</name>&lt;&lt;
"---- Part 1: get an address ----
"</expr_stmt>
<expr_stmt><expr><name>ptr</name> = &amp;<name>x</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "ptr now holds the address: " &lt;&lt;
<name>ptr</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>

</function>"/>
</html>
```cpp
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>

<comment type="block">/**** Part 2: set a dereferenced value ****/</comment>
<expr_stmt><expr><name>cout</name>&lt;&lt;"---- Part 2: dereferencing ----\n"</expr>;</expr_stmt>
<expr_stmt><expr>*<name>ptr</name> = 55</expr>;</expr_stmt>

<comment type="block">/* YOUR CODE HERE */</comment>
<expr_stmt><expr><name>cout</name>&lt;&lt;"x is: " &lt;&lt; <name>x</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>

<comment type="block">/**** Part 3: alias an array ****/</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "---- Part 3: alias an array ----\n"</expr>;</expr_stmt>
<expr_stmt><expr><name>ptr</name> = <name>seq</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name> &lt;&lt; "value of seq (location of array data): " &lt;&lt; <name>seq</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name>&lt;&lt;"value of ptr (where ptr points): " &lt;&lt; <name>ptr</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>
<expr_stmt><expr><call><name>print_array</name><argument_list>(<argument><expr><name>ptr</name></expr></argument>,<argument><expr><name>SEQUENCE_LENGTH</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>

<comment type="block">/**** Part 4: pointer arithmetic ****/</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "---- Part 4: pointer arithmetic ----\n"</expr>;</expr_stmt>
<expr_stmt><expr><call><name>print_address_and_value</name><argument_list>(<argument><expr><name>element</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
<expr_stmt><expr><call><name>print_address_and_value</name><argument_list>(<argument><expr><name>last</name></expr></argument>)</argument_list></call></expr>;</expr_stmt>
```

137
```cpp
<expr_stmt><expr><name>element</name>++</expr>;</expr_stmt>
</block></while>
<expr_stmt><expr><name>cout</name>&lt;&lt;<name>endl</name></expr>;</expr_stmt>

/** Part 5: const ptr vs. ptr to const **/ 
<expr_stmt><expr><name>cout</name>&lt;&lt; "---- Part 5: pointers and constants ---- 
<name>x</name>&lt;&lt; <name>endl</name>
<decl_stmt><decl><type><name>int</name> *<name>const</name></type>
<name>annoying_ptr</name> =<init>&amp;<name>x</name></init></decl>;</decl_stmt>   <comment type="line">// modify this line</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "x (before) is: " &lt;&lt; <name>x</name>&lt;&lt; <name>endl</name>
<expr_stmt><expr><call><name>print_address_and_value</name><argument_list>(<argument><expr><name>annoying_ptr</name></expr></argument>)</argument_list></call></expr_stmt>; 
<expr_stmt><expr>*<name>annoying_ptr</name> = 20</expr_stmt>   <comment type="line">// uncomment this line</comment>
<expr_stmt><expr><name>cout</name> &lt;&lt; "x (after) is: " &lt;&lt; <name>x</name>&lt;&lt; <name>endl</name>
<expr_stmt><expr><call><name>print_address_and_value</name><argument_list>(<argument><expr><name>annoying_ptr</name></expr></argument>)</argument_list></call></expr_stmt>;

/** Part 6: dynamic allocation **/ 
<expr_stmt><expr><name>cout</name> &lt;&lt; "---- Part 6: dynamic allocation ---- 
<expr_stmt><expr><name>cin</name> &gt;&gt; <name>x</name></expr>;
<comment type="line">// demonstrate dynamic allocation:</comment>
<expr_stmt><expr><name>ptr</name> = new <name><name>int</name><index>[<expr><name>x</name></expr>]</name></expr>;
<for>for(<init><decl><type><name>int</name></type>
<name>i</name>=<init><expr>0</expr></init></decl>;</init><condition><expr><name>i</name>&lt;<name>x</name></expr>;</condition><incr><expr>++<name>i</name></expr></incr> ) <block>{{
<expr_stmt><expr><name><name>ptr</name><index>[<expr><name>i</name></expr>]</name> = <name>i</name></expr>;
}}
<expr_stmt><expr><call><name>print_array</name><argument_list>(<argument><expr><name>ptr</name></expr>,
<argument><expr><name>x</name></expr></argument>)</argument_list></call></expr_stmt>;

<comment type="line">// demonstrate dynamic deallocation:</comment>
```
<expr_stmt><expr>delete<index>[]</index> <name>ptr</name></expr>;</expr_stmt>

$return<expr>0</expr>;</return>
</block></function>

<comment type="line">\ //////////////////////////////////</comment>
<comment type="line">// utility functions. do not modify.</comment>
<function><type><name>void</name></type> <name>print_address_and_value</name><parameter_list>(<param><decl><type><name>const</name> <name>int</name>*<name>p</name></decl></param>)</parameter_list> <block>{
  <expr_stmt><expr><name>cout</name> &lt;&lt; <name>p</name> &lt;&lt; " : " &lt;&lt; *<name>p</name> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
}</block></function>

<function><type><name>void</name></type> <name>print_array</name><parameter_list>(<param><decl><type><name>int</name></type> <name>ar</name><index>[]</index></decl></param>, <param><decl><type><name>int</name></type> <name>len</name></decl></param>)</parameter_list> <block>{
  <for><init><decl><type><name>int</name></type> <name>i</name>=<init><expr>0</expr></init></decl>;</init><condition><expr><name>i</name>&lt;<name>len</name></expr></condition><incr><expr>++<name>i</name></expr></incr><block>{
    <expr_stmt><expr><name>cout</name> &lt;&lt; " ", <expr><name>ar</name><index>[<expr><name>i</name></expr>]</expr> &lt;&lt; <name>endl</name></expr>;</expr_stmt>
  }
};
</for>
</block></function>
Figure A.23 Lab 12 RelaxNG

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    ns="http://www.sdml.info/srcML/src" >
    <include href="../../src/includes/cpp-core.rng"/>
    <start>
        <element name="unit">
            <optional><attribute name="language"/></optional>
            <optional><attribute name="filename"/></optional>
            <oneOrMore>
                <choice>
                    <text/>
                    <ref name="comment"/>
                    <ref name="iostream"/>
                    <ref name="iomanip"/>
                    <ref name="cmath"/>
                    <ref name="string"/>
                </choice>
            </oneOrMore>
            <optional><text/></optional>
            <ref name="using"/>
            <optional><text/></optional>
            <!-- Function prototypes -->
            <ref name="comment"/>
            <ref name="function_declaration"/>
            <ref name="function_declaration"/>
            <!-- Main function -->
            <optional><ref name="comment"/></optional>
            <element name="function">
                <element name="type">
                    <element name="name">
                        <text/>
                    </element>
                </element>
                <text/>
                <element name="name">
                    <text/>
                </element>
            </element>
            <text/>
            <element name="name">
                <text/>
            </element>
            <text/>
            <element name="parameter_list">
                <text/>
            </element>
            <text/>
            <element name="block">
                <oneOrMore>
                    <choice>
                        <text/>
                        <ref name="declarative_statement"/>
                        <ref name="expression_statement"/>
                        <ref name="comment"/>
                    </choice>
                </oneOrMore>
            </element>
        </element>
    </start>
</grammar>