**SPL-1 Project Report, 2018**

**A simple program for parsing and analyzing**

**primitive variables**

**SE-306 : Software Project Lab-1**

Submitted by

***AMMAR NOMANY TANVIR***

**BSSE Roll No: 0904**

**BSSE Session: 2016-2017**

Supervised by

***NADIA NAHAR***

**Lecturer**

**Institute of Information Technology**

**University of Dhaka**

****

**Institute of Information Technology**

**University of Dhaka**

Date: 27-05-2016

**Table of Contents**

1.Introduction…………………………………………………….….……………..….03

1.1.Background Study…………………………………………….………………..… 03

* + 1. Memory Layout of C variables ……………………………………….………03

1.1.2 AspectJ ………………………………………………………………………….04

* 1. Challenges……………………………………….………………………...…… 06

2. Project Overview…………………………..….……………….………….………..06

3. User Manual Manual……………………………………………………………….08

3.1 Requirements ……………………………..………………………………………08

3.2 Procedure …………………………………………………………………………09

4. Conclusion………………………………………………………………………….10

References………………………………………………………………………...…..11

**List of Figures**

1. Memory Layout in C………………………………………………………………04

2. Bar Chart…………………………………………………………………………....09

3. Option Window…………………………………………………………………….10

4. Variable Name Window……………………………………………………………10

**1. Introduction**

The purpose of this software project is to get an idea about the variables used in a program, to measure the amount of space they theoretically consume in memory. If total number of variables found in a program of each type can be represented graphically, it becomes easier to get an idea where stack overflow can occur. The target of my project was to implement this idea where I have worked with primitive types of variables only, counted their numbers , total bytes consumed by them and made a bar chart from the number of variables of each type for graphical representation.

* 1. **Background Study**

# 1.1.1.Memory Layout of C Programs

A typical memory representation of C program consists of following sections:

**1. Text Segment:** A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions. As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.

**2. Initialized Data Segment:** Initialized data segment, usually called simply the Data Segment. A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

**3. Uninitialized Data Segment:** Uninitialized data segment (often called BSS segment) starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

**4. Stack:** The stack area traditionally adjoined the heap area and grew the opposite direction which is downwards. Stack, where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn’t interfere with the variables from another instance of the function.

**5. Heap:** Heap is the segment where dynamic memory allocation usually takes place.It grows upwards.The heap area begins at the end of the BSS segment and grows to larger addresses from there.The Heap area is managed by malloc, realloc, and free. The Heap area is shared by all shared libraries and dynamically loaded modules in a process.



Figure 1: Memory Layout in C

**1.1.2. AspectJ**

**Overview of AspectJ**

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code without modification of the code itself. Instead, we declare separately which code is to modify.

AspectJ implements both concerns using extensions of Java programming language.

**Aspect Creation**

AspectJ provides an implementation of AOP and has **three core concepts:**

* Join Point
* Pointcut
* Advice

In order to understand these concepts, following definitions are introduced:

* **Aspect**: A modularization of a concern that cuts across multiple objects. Each aspect focuses on a specific crosscutting functionality.

**aspect** VisitAspect {

**pointcut** set() : **execution**(\* **set**\*(..) ) && **this**(Point);

**after** () : **set**() {

Display.update();

}

}

* **Join point**: A point during the execution of a script, such as the execution of a method or property access, well-defined moments in the execution of a program, like method call, object instantiation, or variable access.
* **Advice**: Action taken by an aspect at a particular join point.

**after** () : **set**() {

Display.update();

}

* **Pointcut**: A regular expression that matches join points. An advice is associated with a pointcut expression and runs at any join point that matches the pointcut.

**pointcut** set() : **execution**(\* **set**\*(..) ) && **this**(Point);

**1.2 Challenges**

I have come across several challenges during my project. Some are as follows:

* The main challenge to complete this project was to learn the basics of AspectJ and using this extension in a java program. Learning and implementing aspects were difficult.
* Learning Threads to implement static and dynamic analysis in the same project and implementing it was quite hard.
* Making a bar chart from the counter of each type variables using API JFreeChart.
* Using java swing for output of variable names.
* At the middle of my project, I came to know that aspects can be created using AspectJ on defined methods, constructors, field and instance variables (i.e. these can be analyzed dynamically) but not on local variables. So, I had to do static analysis of that program to trace all the local variables.
* Give different name for every compressed file related to algorithm name was another challenge. For example if the file name is XXX.txt then the compressed file from this tool is name like XXX\_huff.txt, XXX\_rle.txt, XXX\_lzw.txt and XXX\_lz77.txt. thus user could easily recognize which file is compressed by which algorithm.
* Finally merging all the classes was another challenge.

**2. Project Overview**

There are total 8 classes in my project including main class. The main class is named RunThreads.java. In this class, I have used threads which runs two vital classes called ReadIt.java (it is the program which is being analyzed in this project) and Main.java (it is the most vital class because it calls another classes).

All the Classes and their methods are listed below:

**Class : RunThreads.java**

+Main()

**Class: Main.java**

+run()

**Class : ReadIt.java** [ code for analysis in this project]

+run(): void

+Add(): int

+CircleArea(double): double

+Subtraction(int, int): int

**Class: VarNameCollector.java**

+PushInRespectiveStack(String, int):void

+ExtractVariable(Stack<String>, int): void

+PrintVariables(): void

**Class: LocalVariableCounting.java**

+PushInRespectiveStack(String, int):void

+ExtractVariable(Stack<String>, int): void

+PrintVariables(): void

**Class:BarChart.java**

+main()

+createDataSet()

**Class:ShowVariable.java**

+main()

+initialize()

**Aspect : ClassVariableAndParameterCounting.aj**

+pointcut CountInt()

+pointcut CountDouble()

+pointcut CountFloat()

+pointcut CountChar()

+pointcut CountBoolean()

+pointcut CountString()

RunThreads.java

Thread

Thread

ReadIt.java Main.java

ClassVariableAndParameterCounting.aj

VarNameCollector.java ShowVariable.java

BarChart.java

LocalVariableCounting.java

BarChart.java

Diagram 1: Class relationships.

**3**. **User Manual**

**3.1. Requirements**

**Software**

The project was developed in Java, Java Runtime Environment (JRE) 1.7.0 ( 32 bit / 64 bit version ) or higher versions. Must have supported AspectJ version according to platform. Eclipse or Netbeans IDE with installed AspectJ plugin is preferable.

**Hardware**

This project can run on all PCs operated by Windows or Linux systems having above mentioned software configuration.

**3.2. Procedure**

This project doesn’t require hardest things. Running procedure is quite easy. Here is the process of running it in Eclipse IDE.

Firstly, you have to move to RunThreads.java file and select run button. It will make a window to appear asking for selecting whether to run as AspectJ project or Java project.You have to select run as AspectJ project and next click on finsh button.

After running this project, you will see the bar chart of the number of variables detected in the analyzed program. The bar chart will look something like this-

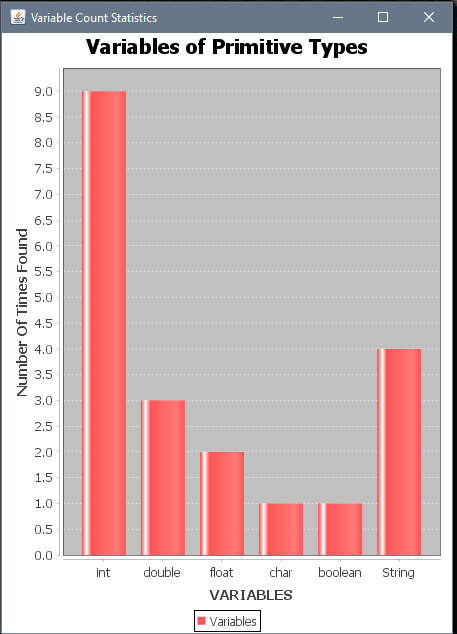


Figure 2: Bar Chart

After the final bar chart, another window will open which asks if you want to see the variable names. Select Yes if you wish to see otherwise No/Cancel button.

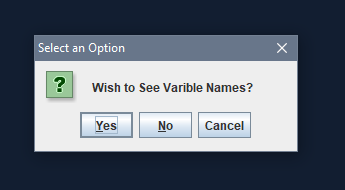


Figure 3: Option Window

In case of selecting Yes button, a window will appear that has six tabs for each primitive type variables. You can select any tab to see variables of that type.

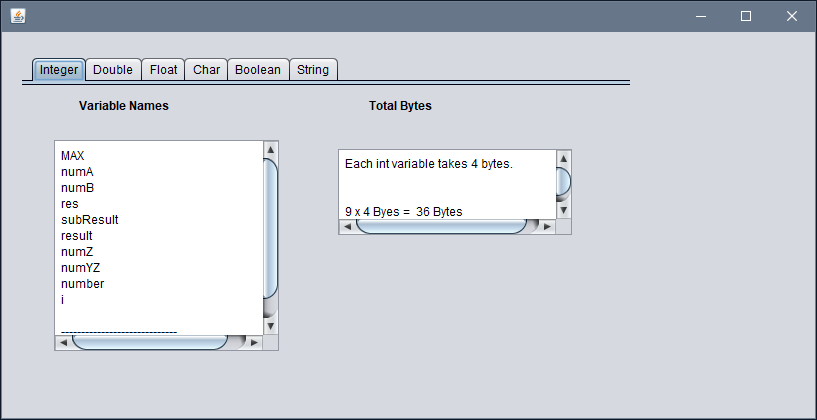
****

Figure 4: Variable Name Window

**4. Conclusion**

I have learned a lot through this project like concept and implementation of AspectJ , memory allocation of variables, Threads in Java and also how to use the JFreeChart API to represent statistical charts and diagrams.

I have used aspects for dynamic analysis of class/field variables, constructor/method parameters and static analysis for tracing local variables.

**References**

[1] <https://www.geeksforgeeks.org/memory-layout-of-c-program/>, last accessed on 28/05/2018.

[2] <https://www.eclipse.org/aspectj/doc/released/progguide/starting.html> - Getting Started With AspectJ, last accessed on 28/05/2018.

[3] <https://www.eclipse.org/aspectj/doc/released/progguide/language.html> - The AspectJ Language, last accessed on 28/05/2018.

[4] <http://www.baeldung.com/aspectj> – Intro to AspectJ, last accessed on 28/05/2018.

[5] **GitHub Link:**

<https://github.com/antanvir/spl/tree/master/SPL-1>