**Error! Unknown document property name.**

**Error! Unknown document property name.**

High Level Design & Low Level Design

The purpose of this document is to provide a template for documenting both HLD & LLD.

**Document Control :**

| **Project Revision History** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |

[**1. Introduction 3**](#_heading=h.3znysh7)

[1.1. Intended Audience 3](#_heading=h.2et92p0)

[1.2. Acronyms/Abbreviations 3](#_heading=h.tyjcwt)

[1.3. Project Purpose 3](#_heading=h.3dy6vkm)

[1.4. Key Project Objectives 3](#_heading=h.1t3h5sf)

[1.5. Project Scope and Limitation 3](#_heading=h.4d34og8)

[*1.5.1. In Scope 3*](#_heading=h.2s8eyo1)

[*1.5.2. Out of scope 3*](#_heading=h.17dp8vu)

[1.6. Functional Overview 3](#_heading=h.3rdcrjn)

[1.7. Assumptions, Dependencies & Constraints 3](#_heading=h.26in1rg)

[1.8. Risks 3](#_heading=h.lnxbz9)

[**2. Design Overview 3**](#_heading=h.35nkun2)

[2.1. Design Objectives 3](#_heading=h.1ksv4uv)

[*2.1.1. Recommended Architecture 3*](#_heading=h.44sinio)

[2.2. Architectural Strategies 3](#_heading=h.2jxsxqh)

[*2.2.1. Design Alternative 3*](#_heading=h.z337ya)

[*2.2.2. Reuse of Existing Common Services/Utilities 3*](#_heading=h.3j2qqm3)

[*2.2.3. Creation of New Common Services/Utilities 3*](#_heading=h.1y810tw)

[*2.2.4. User Interface Paradigms 3*](#_heading=h.4i7ojhp)

[*2.2.5. System Interface Paradigms 3*](#_heading=h.2xcytpi)

[*2.2.6. Error Detection / Exceptional Handling 3*](#_heading=h.1ci93xb)

[*2.2.7. Memory Management 3*](#_heading=h.3whwml4)

[*2.2.8. Performance 3*](#_heading=h.2bn6wsx)

[*2.2.9. Security 3*](#_heading=h.qsh70q)

[*2.2.10. Concurrency and Synchronization 3*](#_heading=h.3as4poj)

[*2.2.11. Housekeeping and Maintenance 3*](#_heading=h.1pxezwc)

[**3. System Architecture 3**](#_heading=h.49x2ik5)

[3.1. System Architecture Diagram. (Not Necessary) 3](#_heading=h.2p2csry)

[3.2. System Use-Cases 3](#_heading=h.147n2zr)

[3.3. Subsystem Architecture 3](#_heading=h.3o7alnk)

[3.4. System Interfaces 3](#_heading=h.23ckvvd)

[*3.4.1. Internal Interfaces 3*](#_heading=h.ihv636)

[*3.4.2. External Interfaces 3*](#_heading=h.32hioqz)

[**4. Detailed System Design 3**](#_heading=h.1hmsyys)

[4.1. Key Entities 3](#_heading=h.41mghml)

[4.2. Detailed-Level Database Design 3](#_heading=h.2grqrue)

[*4.2.1. Data Mapping Information 3*](#_heading=h.vx1227)

[*4.2.2. Data Conversion 3*](#_heading=h.3fwokq0)

[4.3. Archival and retention requirements 3](#_heading=h.1v1yuxt)

[4.4. Disaster and Failure Recovery 3](#_heading=h.4f1mdlm)

[4.5. Business Process workflow 3](#_heading=h.2u6wntf)

[4.6. Business Process Modeling and Management (as applicable) 3](#_heading=h.19c6y18)

[4.7. Business Logic 3](#_heading=h.3tbugp1)

[4.8. Variables 3](#_heading=h.28h4qwu)

[4.9. Activity / Class Diagrams (as applicable) 3](#_heading=h.nmf14n)

[4.10. Data Migration 3](#_heading=h.37m2jsg)

[*4.10.1. Architectural Representation 3*](#_heading=h.1mrcu09)

[*4.10.2. Architectural Goals and Constraints 3*](#_heading=h.46r0co2)

[*4.10.3. Logical View 3*](#_heading=h.2lwamvv)

[*4.10.4. Architecturally Significant Design Packages 3*](#_heading=h.111kx3o)

[*4.10.5. Data model 3*](#_heading=h.3l18frh)

[*4.10.6. Deployment View 3*](#_heading=h.1egqt2p)

[**5. Environment Description 3**](#_heading=h.3ygebqi)

[5.1. Time Zone Support 3](#_heading=h.2dlolyb)

[5.2. Language Support 3](#_heading=h.sqyw64)

[5.3. User Desktop Requirements 3](#_heading=h.3cqmetx)

[5.4. Server-Side Requirements 3](#_heading=h.1rvwp1q)

[*5.4.1. Deployment Considerations 3*](#_heading=h.4bvk7pj)

[*5.4.2. Application Server Disk Space 3*](#_heading=h.2r0uhxc)

[*5.4.3. Database Server Disk Space 3*](#_heading=h.1664s55)

[*5.4.4. Integration Requirements 3*](#_heading=h.3q5sasy)

[*5.4.5. Jobs 3*](#_heading=h.25b2l0r)

[*5.4.6. Network 3*](#_heading=h.kgcv8k)

[*5.4.7. Others 3*](#_heading=h.34g0dwd)

[5.5. Configuration 3](#_heading=h.1jlao46)

[*5.5.1. Operating System 3*](#_heading=h.43ky6rz)

[*5.5.2. Database 3*](#_heading=h.2iq8gzs)

[*5.5.3. Network 3*](#_heading=h.xvir7l)

[*5.5.4. Desktop 3*](#_heading=h.3hv69ve)

[**6. References 3**](#_heading=h.1x0gk37)

[**7. Appendix 3**](#_heading=h.4h042r0)

# 

# Introduction

Static Dynamic analysis is the testing and evaluation of an application by examining the code at run time and executing the application. Many software defects that cause memory and threading errors can be detected both dynamically and statically.

A static analysis tool is a program that analyzes code or other artifacts without actually executing them. This can be useful for identifying potential issues or vulnerabilities in code, such as syntax errors, security vulnerabilities, or code that does not adhere to coding standards.And dynamic analysis tool, on the other hand, is a program that analyzes code or other artifacts by executing them and observing their behavior at runtime. This can be useful for identifying issues.

## 1.1 Intended Audience

Helps to find out the errors in the programs by using different tools such as Valgrind, Splint and Makefile.

## Acronyms/Abbreviations

| SDAT | Static Dynamic Analyser Tool |
| --- | --- |
| ti | Time |
| str | String |
| sp | Splint |
| val\_tool | Valgrind |
| exe\_file | Executable File |
| vl | To create valgrind lock file |
| d | Date |
| cmd | Command |

## Project Purpose

The purpose of a static dynamic analysis tool is to analyze code or other artifacts without actually executing them, in order to identify potential issues or vulnerabilities.And dynamic analysis tool is to analyze code or other artifacts by executing them and observing their behavior at runtime.

Both static and dynamic analysis tools can be used to improve the quality and security of software. The choice of which type of tool to use depends on the specific goals and needs of the software development project

## Key Project Objectives

Finding syntax errors or other coding issues that could cause the software to malfunction.

Ensuring that the code adheres to coding standards and best practices Improving the maintainability of the code by identifying areas that are difficult to understand or modifyIdentifying performance issues or bottlenecks that could affect the usability or scalability of the software.

Finding runtime errors or other issues that occur during the execution of the software.Testing the functionality of the software to ensure that it behaves as expected.Identifying potential security vulnerabilities that may not be apparent from static analysis. We have used valgrind, Splint and Makefile to check warning, errors memory leak, reduse compilation time.

## Project Scope and Limitation

Project scope and limitation by using static dynamic tool analyser can help to improve the quality and security of the software by identifying and addressing potential problems before the software is deployed.dynamic analysis tool is to identify issues or vulnerabilities in code or other artifacts by executing them and observing their behavior at runtime.

### 1.5.1 In Scope

* + Supported programming languages: The tool may support analysis of code written in one or more programming languages.
  + Analysis capabilities: The tool may be able to perform various types of analysis, such as identifying potential security vulnerabilities, detecting code smells or other quality issues, or enforcing coding standards.
  + Integration with development workflow: The tool may be integrated into the development environment or build process, allowing developers to run the analysis as part of their normal workflow.

### 1.5.2 Out of scope

* + Limited scope: Static analysis tools can only analyze the code that is available to them, and cannot take into account any external factors or dynamic behavior. This means they may not be able to detect certain types of issues that can only be identified through dynamic analysis.
  + Limited understanding of context: Static analysis tools do not have the ability to understand the context in which the code is being used, which can lead to them flagging issues that are actually acceptable in the specific context of the code.
  + Limited ability to fix issues: While static analysis tools can identify potential issues in the code, they cannot automatically fix them. Developers still need to manually review and fix any issues that are identified.
  + Performance impact: Depending on the size and complexity of the codebase, running a static analysis tool can be resource-intensive and may impact the performance of the development environment.

## Functional Overview

* Valgrind works by executing the program and monitoring its memory access patterns, and can identify a wide range of issues such as memory leaks, buffer overflows, use of uninitialized variables, and invalid memory accesses.
* SPLINT checks for a variety of issues such as type errors, null dereferences, and uninitialized variables, and can also enforce coding standards and perform design rule checking.
* A Makefile is a file used to build and manage projects. It specifies the build rules for the project, including how to compile and link the source code, and can also include other tasks such as running tests or generating documentation.

## Assumptions, Dependencies & Constraints

**Software-** Linux.

**Operating System-** Ubuntu.

## Risks

* Security vulnerabilities: Static analysis tools can help to identify potential security vulnerabilities in the code, such as buffer overflows or use of unsafe functions, which can be exploited by attackers.
* Quality and reliability issues: Static analysis tools can help to identify issues such as memory leaks or use of uninitialized variables, which can cause the program to crash or behave unexpectedly.
* Non-compliant code: Static analysis tools can be configured to enforce coding standards and identify code that does not comply with these standards. This can help to ensure that the code is maintainable and easy to understand.
* Performance issues: Static analysis tools can help to identify code that may be inefficient or have a negative impact on performance, allowing developers to optimize it before it becomes a problem.

# Design Overview

This project is designed by using three tools. Such as Valgrind, Splint and Makefile.

**Valgrind** is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

**Splint** is a tool for statically checking C programs for security vulnerabilities and coding mistakes. With minimal effort, Splint can be used as a better lint. If additional effort is invested adding annotations to programs, Splint can perform stronger checking than can be done by any standard lint.

**Makefile** is a program building tool which runs on Unix, Linux, and their flavors. It aids in simplifying building program executables that may need various modules. To determine how the modules need to be compiled or recompiled together, make takes the help of user-defined makefiles.

## Design Objectives

The objective of the Static Dynamic Analyzer tool is to find out the errors with the Splint tool and find memory leaks with Valgrind. Makefile is used to compile the C files and find out the errors.

### 2.1.1 Recommended Architecture

In this project we have used different tools to fulfill the requirements such as Valgrind and Splint to find out the errors. Makefile works to compile C files and find out the errors.

## Architectural Strategies

In this Project the main three requirements are that they performed Splint ,Valgrind and Makefile. So we created three options where each option is working for respectively Splint, Valgrind and Makefile *.*

**Design Alternative**

Not Applicable.

### Reuse of Existing Common Services/Utilities

Not Applicable.

### Creation of New Common Services/Utilities

Not Applicable*.*

### User Interface Paradigms

It is working as tools which work as both Static and Dynamic.Like in static we used Splint tool and in Dynamic we used Valgrind and Makefile.

### System Interface Paradigms

In this project there are three tools which we used the Splint tool is work as to input the File name then it generates the report and stores the output file with Timestamp.Valgrind is work as to input the object file name then it generates the report and store in a file with Timestamp. Makefile works to compile the c-files and detect the errors and find out the errors.

### Error Detection / Exceptional Handling

Error detection and exception handling in static and dynamic analysis tools involve identifying and addressing errors or unexpected conditions that may occur during the analysis process.Static analysis tools, also known as static code analysis or static program analysis, are used to analyze the source code of a program without executing it.And Dynamic analysis tools, also known as dynamic code analysis or dynamic program analysis, are used to analyze the behavior of a program while it is executing.

### Memory Management

Memory management in static analysis tools generally involves managing the memory resources used by the analysis algorithms, such as allocating and freeing memory for data structures and temporary variables.And in dynamic analysis tools generally involve monitoring the program's use of memory resources, such as measuring heap and stack usage, detecting memory leaks, and identifying memory-related performance issues.

### Performance

Performance in Static Dynamic Tool Analysis is the static analysis tools can be affected by the size and complexity of the code being analyzed, as well as the efficiency of the analysis algorithms used.And in dynamic analysis tools can be affected by the size and complexity of the program being analyzed, as well as the overhead of the analysis process on the program's execution. Dynamic analysis tools that can analyze the program with minimal impact on its performance are considered to have good performance.

### Security

In Static Dynamic Tool Analysis the security in static analysis tools can be improved by using advanced analysis algorithms that can detect a wide range of vulnerabilities and by incorporating security best practices and guidelines into the analysis process.and Dynamic analysis tools can be improved by using advanced monitoring and instrumentation techniques that can detect a wide range of vulnerabilities and by incorporating security best practices and guidelines into the analysis process.

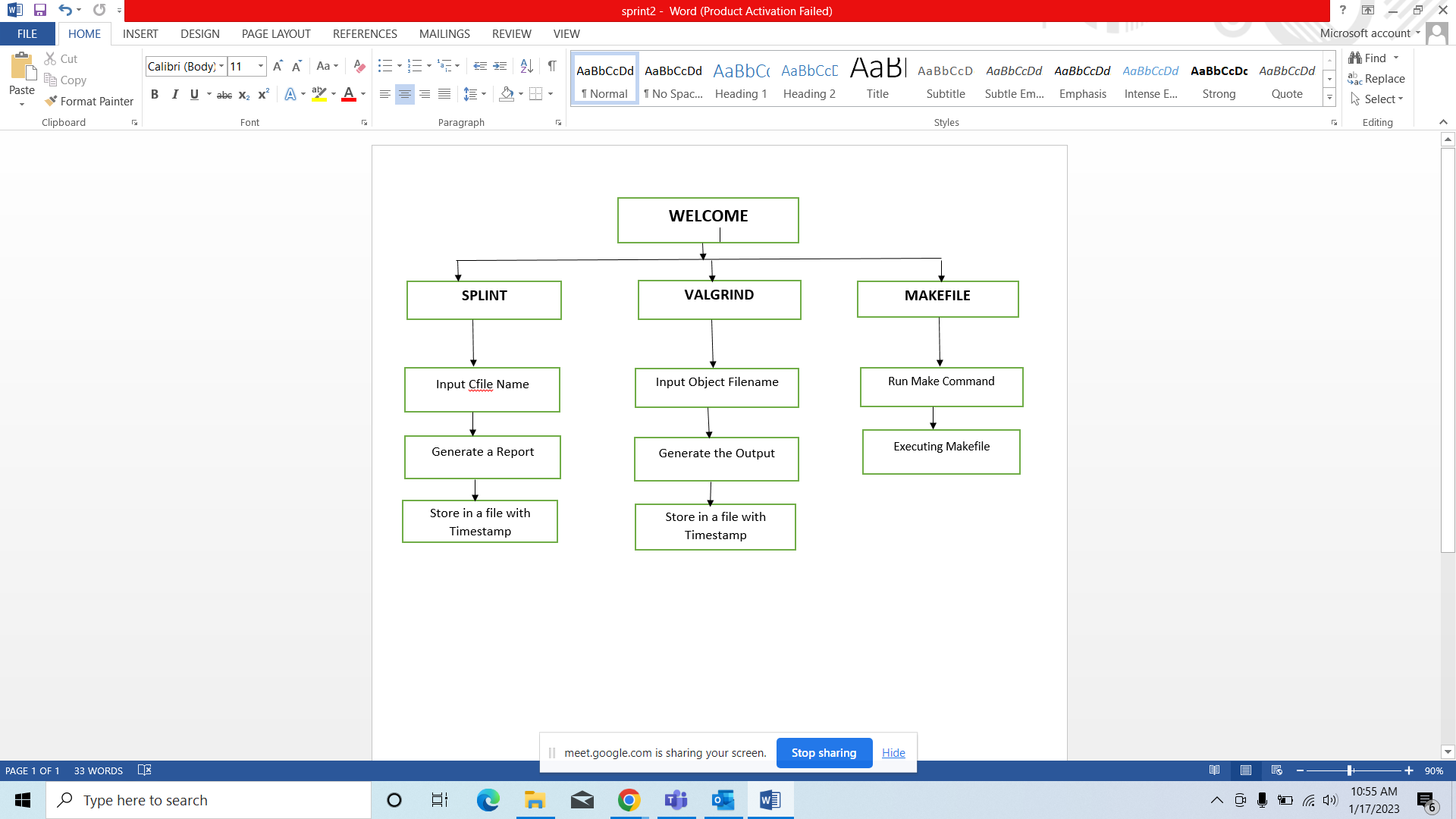
### Concurrency and Synchronization

Static Dynamic Tool Analyzer (SDTA), concurrency and synchronization would likely be used to ensure that multiple threads or processes that are analyzing the same code or data do not interfere with each other, and that the results of the analysis are accurate and consistent.

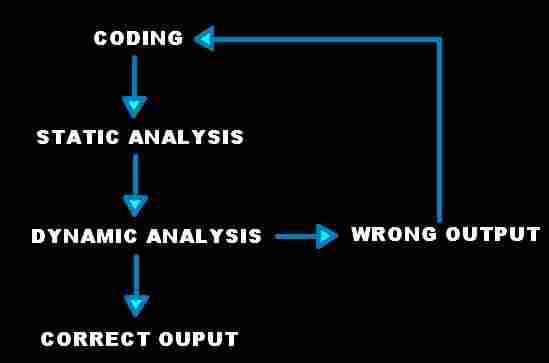
### Housekeeping and Maintenance

Not Applicable.

# System Architecture

**

## System Use-Cases

**

## Subsystem Architecture

To create the Static Dynamic analyser tool we have used three tools such as Valgrind, Splint and Makefile.This three tools have different functionalities.

Splint is a programming tool for statically checking programs for security vulnerabilities and coding mistakes.

Valgrind is a tool for finding memory access errors to heap memory that is dynamically allocated with new or malloc in the programming.

Makefile is a set of commands with variable names and targets to create object files and to remove them. In a single makefile we can create multiple targets to compile and remove objects.

**3.3 System Interfaces**

### Internal Interfaces

As an Internal Interface we are using Ubuntu Distribution. It is an operating system that is made up of a collection of software based on Linux Kernel or you can say that distribution contains the Linux Kernel and Supporting libraries and softwares.

### External Interfaces

Desktop or Linux Operating System.

# Detailed System Design

There are three parts in a project: splint,valgrind and makefile. and we are performing splint on .c file and we are performing valgrind on object file of .c file.

## Key Entities

In this project there are three tools. : 1. Splint

2. Valgrind

3. Makefile

and we are doing operations on the .c file and its compiled files.

## Detailed-Level Database Design

In this project we are using redirect for writing the console output to the file of splint operation output and the file name is splint.txt.

And a log file for writing the console output to the file for valgrind.

### Data Mapping Information

*[The detailed data mapping information has to be documented here. Describe in detail the requirements of data mapping, Data Models to be mapped, Integration details etc. including*

[*Data transformation*](http://en.wikipedia.org/wiki/Data_transformation) *or* [*data mediation*](http://en.wikipedia.org/wiki/Data_mediation) *between a data source and a destination*

*Identification of data relationships as part of data lineage analysis*

*Discovery of hidden and sensitive data, such as data masking.*

[*Consolidation*](http://en.wikipedia.org/w/index.php?title=Data_consolidation&action=edit&redlink=1) *of multiple databases into a single database and identifying redundant columns of data for consolidation or elimination.]*

### Data Conversion

Not applicable.

## Archival and retention requirements

*Not applicable*

## Disaster and Failure Recovery

*Not applicable*

## Business Process workflow

Not Applicable.

## Business Process Modeling and Management (as applicable

Not Applicable.

## Business Logic

Not Applicable.

## Variables

We have used variables, naming conventions in our project are listed below:

**1.int** – ch,f

**2.String –** take\_ti, report\_s, report\_v, val\_cmd, sprint\_cmd, exe\_file , vl , sp, d.

**3. char** *–* buffer.

## Activity / Class Diagrams (as applicable)

## Data Migration

Not applicable.

### Architectural Representation

Not Applicable.

### Architectural Goals and Constraints

Not Applicable.

### Logical View

Not Applicable.

### Architecturally Significant Design Packages

Not Applicable.

### Data model

*[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]*

### Deployment View

*[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software, and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the* ***Process View*** *onto the physical nodes.]*

# Environment Description

We are using the Linux Operating System.

## Time Zone Support

Supports all time zones.

## Language Support

In our project we have used C++ programming language for coding purposes and a .c file used to perform splint.

## User Desktop Requirements

Windows : 7 or above

Processor: Minimum 1GHz and more

Hard Drive: 32GB and more

Memory(RAM): Minimum 1GB and more

## Server-Side Requirements

Not Applicable.

### Deployment Considerations

Not Applicable.

### Application Server Disk Space

Not Applicable.

### Database Server Disk Space

Not Applicable.

### Integration Requirements

In this project we are keeping all the .file and .cpp files in one directory and while running the program we are passing .c file path while performing splint and passing object file of program for performing valgrind.

### Jobs

Not Applicable.

### Network

Not Applicable.

### Others

Not Applicable.

## Configuration

Not Applicable.

### Operating System

Windows : 7 or above

Processor:Minimum 1GHz and more

Hard Drive: 32GB and more

Memory(RAM): Minimum 1GB and mor*e*

### Database

Not Applicable.

### Network

Not Applicable.

### Desktop

We are using Linux Ubuntu 3.2

# References

1) <https://www.geeksforgeeks.org/system-call-in-c/>

2) h[ttps://www.tutorialspoint.com/cpp\_standard\_library/cpp\_string\_c\_str.htm](https://www.tutorialspoint.com/cpp_standard_library/cpp_string_c_str.htm)

3)<https://askubuntu.com/questions/420981/how-do-i-save-terminal-o> utput-to-a-file

# 7. Appendix

Valgrind

Makefile

Splint

**Change Log**

| **QMS Template Version Control (Maintained by QA)** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
| 28-May-2015 | 1.0 | QA Team | | Initial Version | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |