Evaluating Protocol Parser Performance

Henning Schulzrinne
Department of Computer Science
Columbia University
NY 10027
hgs@cs.columbia.edu

Aarthi Venkataramanan
Department of Computer Science
Columbia University
NY 10027
av2410@columbia.edu

Abstract

Internet messages are one of the most prevalent forms of communication today, understanding optimal ways to store and parse this information can help save time. In this study, we attempt to compare parsers for Internet messages based on the underlying representation of the input they work on. The formats considered for this purpose are text as per RFC 5322 [1,2], XML and Binary.

General Terms
Parsers, Performance, Comparison

Keywords
Internet messages, xml encoding, binary encoding, tlvs, rfc 5322

Related Work

Documented work for parsing electronic mail headers is sparse and relatively less known. GMime [3] is a C/C++ library for parsing MIME messages but was difficult to apply the API (I/O classes) to current work to retrieve header information and so could not be used.

1 Introduction

Internet messages consist of a set of header fields (header section) followed by an optional body. In this experiment, the focus is on the header section of Internet messages. The aim of this parser comparison study is to analyze the differences in the implementation and execution speeds that can be achieved with storing the header section of Internet messages in three primary forms:
1) The original Internet message format prescribed and defined by RFC 5322 2) XML data [3,4,5] and 3) as raw binary data.

Parsing in each case is designed to extract information from the e-mail header fields and this information is then stored in a structure that is similar across the three types. All parsers are in C.

E-mail header fields are defined by the RFC to be lines starting with a field name, followed by a colon (:), followed by a field body, and terminated by CRLF [1]. This field body can have printable US ASCII characters and WSP but must not contain CR or LF.Each line is recommended to have no more than 78 characters [1,2]. XML allows for more structuring of the input, and the header fields' data can be stored under relevant tags and attributes. In the binary version a type-length-value encoding is used to present the header fields.

1.1 Outline

The rest of the report is organized as follows – In **Section 2**, the system requirements and environment are described. The actual implementation is discussed in **Section 3**. **Section 4** describes performance evaluation measures and results obtained. Conclusions drawn are listed in **Section 5**.

2 System Requirements

The experiment was performed on the following system:

Operating System: Ubuntu 9.10
Processor: 2.26 GHz Intel Core 2 Duo Processor
Memory: 2 GB 1067 MHz DDR3

Libraries /Compilers
gcc –GNU Compiler Collection [6]
Libxml2 - XML C parser available under MIT license [7,8,9]
(Appendix 2 lists detailed installation instructions)

3 Implementation

The parsers were implemented in C on a Linux platform. Of the three, it was observed that a useful C parsing library that could be adapted and used for Internet messages was available only for XML data, in the form of libxml2 [7]. The binary and RFC parser's functionality had to be built extensively from the ground up.

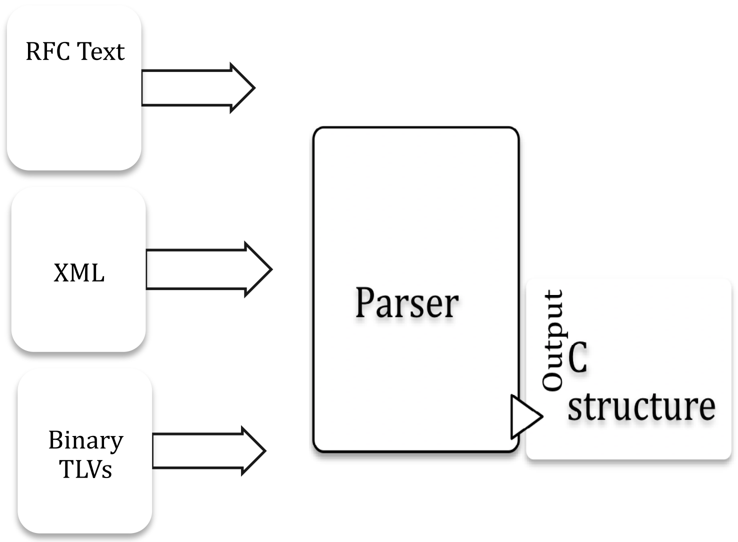


Figure 1: High-level architecture

C struct

The C structure that is the holder for the values extracted after parsing comprises both char pointers and nested structures for variable header fields. (Appendix 1 has definition)

3.1 RFC

The specification [1] for Internet messages as per the Augmented Backus-Naur Form rules was used to define the input (Appendix 3.1) .A finite state machine-like model was used to parse the incoming file. Each character read would then be an input, which determined next state of parser. Validation on the fields was typically performed on values retrieved after parsing.

The major modules in this parser are:
parse_fields: this module performed actual separation of the different components of the header fields(such as via,by,date and from components for the received fields, display name and e-mail address components for the to field ,etc.)
validate_email, validate_fields: In these modules the field values were checked to ensure validity of input.

3.2 XML

Libxml2 libraries [7,8,9] were used to parse the XML document (Appendix 3.2) and fetch the tags and the attributes; <libxml/xmlmemory.h> and <libxml/parser.h> were the headers included for compilation.*xmlDocPtr,xmlNodePtr* and *xmlAttrPtr* pointers to the root,current node(or tag) and its attributes as defined in the parser module of libxml2 [7] were used to achieve desired results.*parse_fields and validate_email* are again the major modules of this parser.

3.3 Binary

A structure representing the type-length-value-input was used to create the binary file [10] as well as read back the values parsed. Types corresponding to the various header fields define in RFC 5322 were arbitrarily assigned unique numbers (mappings described in Appendix 3.3). Structure pointers, size and number of bytes were used as parameters in reading in the values from the binary file.

Challenges in implementation
While the nested inner level structure representation for variable sized headers worked for small sizes, generalizing to work with larger sizes (header fields) was difficult to achieve and produced segmentation faults. Current implementation needs work to fix this issue.

4 Performance Evaluation

In order to compare the efficiency of each parsing method, execution time was measured as recommended by GNU [11] using the system clock.

Parser code was allowed to execute 1000 times to facilitate time measurement and the experiment was repeated 10 times for better accuracy. The average time taken for one execution of the code was then derived through even more repetitions of the process.(Appendix 4 lists detailed results)

A summary of the results obtained is as given below:

Type of input	time taken for one execution/run of the parser (in milliseconds)
RFC txt	0.164
xml file	0.125
bin file	0.033

Table 1 Parser execution times comparison

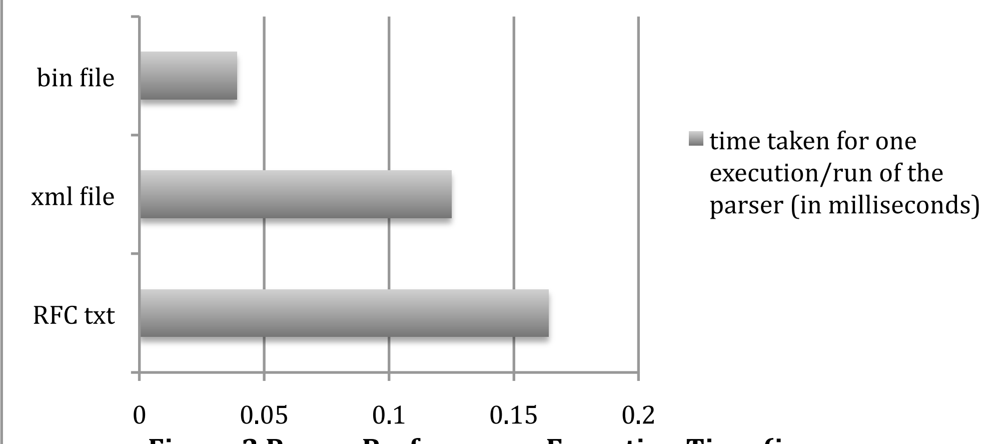


Figure 2 Parser Performance:Execution Time (in milliseconds)

It was seen that the binary format was the fastest to parse, and then followed by the xml file and finally the RFC.The differences in speeds were in fractions of microseconds albeit consistent and hence noticeable.

The results in favor of the binary format may be attributed to the efficiency and structure of the type-length-value format as data can be extracted by in effect reading back the values from the binary file and the workload of the parser is considerably reduced.

As with the XML, the use of XML attributes to separate components of the header fields (for e.g. display names of email addresses could be more naturally represented as an attribute of the corresponding node in xml, separating it from the email address which would be the actual value of the node) is most likely the reason for improved performance over the RFC parser which had to parse out these details as well.

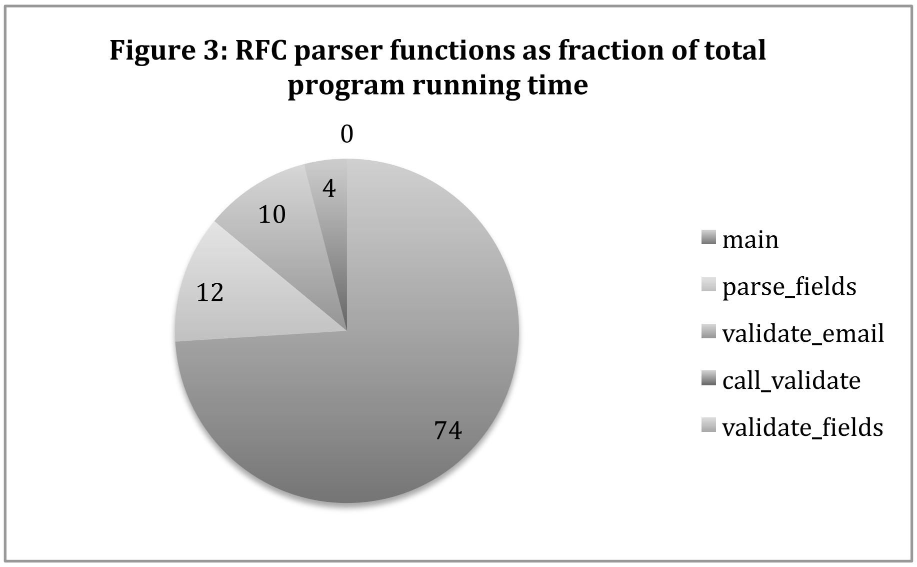
Profiling

gprof [12] a useful UNIX profiling command was used to analyze the time spent in each of the functions involved in the three parsers.(refer to Appendix 5 for steps to run gprof)

The following charts illustrate typical observed behavior.

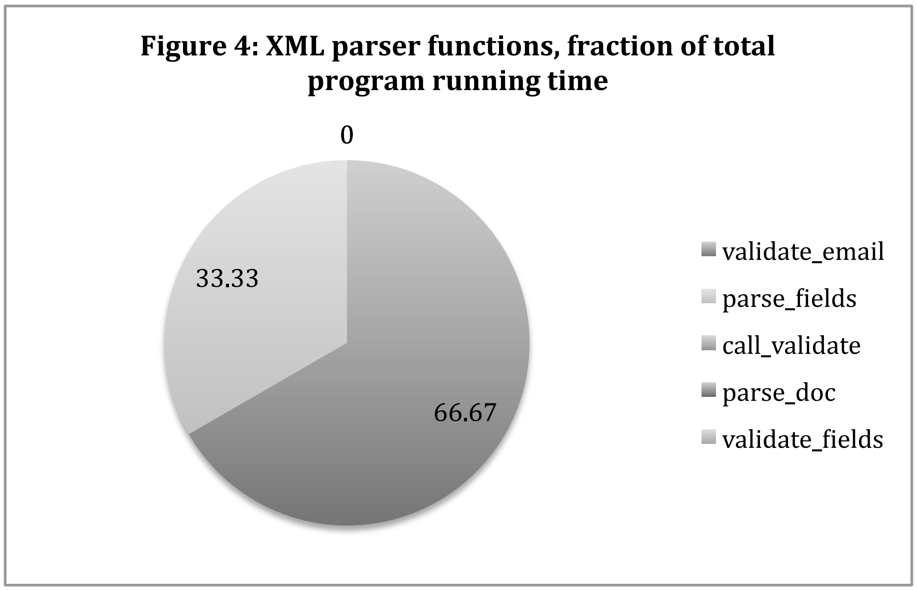
function	% of total program running time
main	74
parse_fields	12
validate_email	10
call_validate	4
validate_fields	0

Table 2 :RFC functions flat profile data



function	% of total program running time
validate_email	66.67
parse_fields	33.33
call_validate	0
parse_doc	0
validate_fields	0

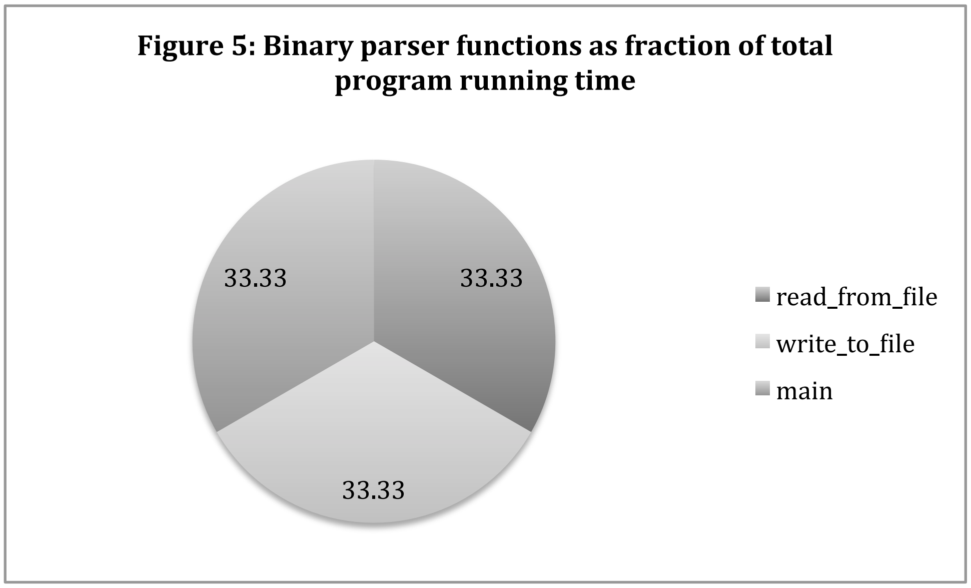
Table 3: XML functions flat profile data



Note:
For some functions in both of the above cases, the % of total program running time was relatively short compared to other functions and hence show on the profile with a value of 0% despite being called in the program as shown by the number of calls made (Source Code attachment lists the actual file that was output by the gprof command with additional details like number of calls, ms per call, etc)

function	% of total program running time
main	33.33
read_from_file	33.33
write_to_file	33.33

Table 5: Binary functions flat profile data



For the purpose of profiling, the write_to_file function was considered however it was not included in measurement of actual execution time of the parser as its function was to create the binary file, which is actually the input for parsing.

5 Conclusion

In this study performed on the same input data and measurement parameters (1000 executions repeated 10 times on the same environment for all 3 parsers) it was found that of the three, parsing was fastest with the binary representation. The XML follows up second, with the RFC taking the longest to parse. The execution times for all 3 parsers were in fractions of milliseconds.

References

[1] Resnick, P., "Internet Message Format", Request For Comments 5322,October 2008
[2] “GMime “,http://spruce.sourceforge.net/gmime/
[3] XML W3C Recommendation , http://www.w3.org/TR/xml/
[4] XML W3C Recommendation, http://www.w3.org/TR/REC-xml/
[5] www.w3schools.com/**xml**/default.asp
[6] “GNU Compiler Collection”,http://gcc.gnu.org/
[7] “The Libxml2 Reference Manual”, http://xmlsoft.org/html/libxml-lib.html
[8] “Libxml tutorial”, http://xmlsoft.org/tutorial/index.html
[9]“Libxml2 download”, http://xmlsoft.org/downloads.html
[10] “Binary I/O”, http://www.gnu.org/software/octave/doc/interpreter/Binary-I_002fO.html#doc%2dfwrite
[11] “CPU Time Inquiry”, http://www.gnu.org/s/libc/manual/html_node/CPU-Time.html
[12] “gprof” http://manpages.ubuntu.com/manpages/hardy/man1/gprof.1.html

Appendix 1
C struct definition

```
struct parse_header{

        char *ptr_recvd_from;
        char *ptr_recvd_via;
        char *ptr_recvd_by;
        char *ptr_recvd_with;
        char *ptr_recvd_id;
        char *ptr_recvd_for_email;
        char *ptr_recvd_for_date;
        char *ptr_resent_date;
        struct resent_from *ptr_resent_from_field;
        struct resent_to *ptr_resent_to_field;
        struct resent_cc *ptr_resent_cc_field;
        struct resent_bcc *ptr_resent_bcc_field;
        char *ptr_resent_message_id;
        char *ptr_orig_date;
        struct from_field *ptr_from_field;
        struct to_field *ptr_to_field;
        struct reply_to_field *ptr_reply_to_field;
        struct cc_field *ptr_cc_field;
        struct bcc_field *ptr_bcc_field;
        char *ptr_message_id;
        char in_reply_to[20][50];
        char references[20][50];
        char *ptr_subject;
        char *ptr_comments;
        char *ptr_keywords;
        char *ptr_received;
        char *ptr_return_path;

}ph;
```

Appendix 2
Setup & Installation

<u>2.1 Installing gcc and libxml libraries</u>

2.1 Installing gcc

```
$ sudo apt-get update
$ sudo apt-get install build-essential
$ gcc –v
$ make -v
```

<u>2.2 Libxml2 installation</u>

After downloading the latest version of libxml2 from [11], these steps should be executed:

```
$ ./configure --prefix=/usr/local/libxml2
$ make
$ sudo make install
```

This displays the message:

```
Libraries have been installed in:
   /usr/local/libxml2/lib
```

Set path by typing:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/libxml2/lib
```

To compile :

```
gcc -I/usr/local/libxml2/include/libxml2 -lxml2  filename.c
./a.out input.xml
```

Appendix 3
Sample Inputs

3.1 RFC Input Sample

```
Return-Path : <local@domain>
Received : from x.y.test by example.net via TCP with ESMTP id ABC12345 for
<mary@example.net>;  21 Nov 1997 10:05:43 -0600
Resent-Date : Fri, 21 Nov 1997 09:55:06 -0600
Resent-From : Zhou Chang <ching@corp.com>
Resent-To : Mona Baker <mbaker@bing.com>, Vera
Braldey<vbdly@hotmail.org>
Resent-Cc : Peter Smith <psmith@gmail.com>, "Jacob Karen"
<jkr@hotmail.org>
Resent-Bcc : Ming Fu <ming@bing.com>, <dring@hotmail.org>
Resent-Message-ID : <8912@bon.com>
Date :Fri, 20 Nov 1997 09:55:06 -0600
From :John Doe <jdoe@machine.example>
Reply-To : <mwng@bing.com>, "Vera Braldey" <vbly@hotmail.org>
To : Mary Baryy <mary@example.net>,sjakes@eee.com,<areva@pearlmail.in>
Cc :Maya@example.net
Bcc :"John Doe Personal Account" <doe@home.example>,"Mason"
<mason@sate.com>,"Kate" <kates@gmail.com>,<harry@jiffy.com>
Message-Id : <1234@machine.example>
In-Reply-To :<4567@pearlmail.in>, <9123@example.net>
References :<4567@pearlmail.in>, <5678@example.net>
Subject :Saying Hello
Comments :Greeting
Keywords :meeting up, general discussion
```

3.2 XML Input Sample

```xml
<internet-message xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:SchemaLocation="schema.xsd">
<trace>
    <return-path> &lt;mary@example.net &gt;</return-path>
    <received from="x.y.test" by="example.net" via="TCP"
with="ESMTP" id="ABC12345" for="&lt;mary@example.net&gt;">
    <date>2002-10-10T17:00:00Z</date>
    </received>
         <received from="node.example" by="x.y.test">
        <date>2002-10-10T16:55:00Z</date>
    </received>
        <resent-date>2002-10-11T16:55:00Z</resent-date>
        <resent-to displayname="Zhou
Chang">&lt;ching@corp.com&gt;</resent-to>
        <resent-cc displayname="Peter
Smith">&lt;psmith@gmail.com&gt;</resent-cc>
        <resent-bcc displayname="Ming
Fu">&lt;ming@bing.com&gt;</resent-bcc>
        <resent-from displayname="Mona
Baker">&lt;mbaker@bing.com&gt;</resent-from>
        <resent-message-id>&lt;8912@bon.com&gt;</resent-message-
id>

</trace>
<orig-date>2002-10-10T16:50:00Z</orig-date>
<from displayname="John Doe">&lt;jdoe@machine.example&gt;</from>
<reply-to displayname="John Doe: Personal
Account">&lt;doe@home.example&gt;</reply-to>
<to displayname="Mary Smith">&lt;mary@example.net&gt;</to>
<cc displayname="Boss">&lt;boss@nil.test&gt;</cc>
<bcc displayname="Mary Smith">&lt;mary@example.net&gt;</bcc>
<message-id>&lt;3456@local.machine.example&gt;</message-id>
<in-reply-to>&lt;1234@example.net&gt;</in-reply-to>
<references>&lt;1234@example.net&gt;</references>
<subject> Saying Hello</subject>
<comments> Greeting</comments>
<keywords> meeting up, general discussion </keywords>
</internet-message>
```

3.3 TLV sample used to create .bin file

input_tlv.txt
```
Type Value
5 <local@domain>
8 Zhou Chang <ching@corp.com>
9  Mona Baker <mbaker@bing.com>
22  "Jacob Karen" <jkr@hotmail.org>
23  Ming Fu <ming@bing.com>
24 <8912@bon.com>
15 Fri, 20 Nov 1997 09:55:06 -060
14 John Doe <jdoe@machine.example>
13 <mwng@bing.com>
10  Mary Baryy <mary@example.net>
11 Maya@example.net
12 "John Doe Personal Account" <doe@home.example>
16 1234@machine.example>
17 <4567@pearlmail.in>
19 Saying Hello
20 Greeting
21 meeting up general discussion
```

mapping types
```
Return-Path 05
Resent-Date 07
Resent-From  08
Resent-To 09
Resent-Cc 22
Resent-Bcc 23
Resent-Message-ID 24
Date 15
From 14
Reply-To 13
To 10
Cc 11
Bcc 12
Message-Id  16
In-Reply-To 17
References 18
Subject 19
Comments 20
Keywords 21
```

Appendix 4

4.1 RFC Detailed Execution Times Result

```
RESULT: Time elapsed or CPU time used: 0.210000 s
RESULT: Time elapsed or CPU time used: 0.150000 s
RESULT: Time elapsed or CPU time used: 0.140000 s
RESULT: Time elapsed or CPU time used: 0.130000 s
RESULT: Time elapsed or CPU time used: 0.130000 s
RESULT: Time elapsed or CPU time used: 0.180000 s
RESULT: Time elapsed or CPU time used: 0.120000 s
RESULT: Time elapsed or CPU time used: 0.170000 s
RESULT: Time elapsed or CPU time used: 0.270000 s
RESULT: Time elapsed or CPU time used: 0.140000 s
TOTAL TIME TO RUN THE CODE 1000 times and repeat the experiment 10 times
=1.640000 s
AVERAGE TIME to run the CODE 1000 times= 0.164000 s
estimated time taken to run the code for one execution( avg time /1000)= 0.164 ms
```

(Note: RESULT: Time elapsed denotes time for a 1000 executions of the code)

4.2 XML Detailed Execution Times Result

```
RESULT: Time elapsed or CPU time used: 0.100000 s
RESULT: Time elapsed or CPU time used: 0.150000 s
RESULT: Time elapsed or CPU time used: 0.150000 s
RESULT: Time elapsed or CPU time used: 0.100000 s
RESULT: Time elapsed or CPU time used: 0.140000 s
RESULT: Time elapsed or CPU time used: 0.100000 s
RESULT: Time elapsed or CPU time used: 0.120000 s
RESULT: Time elapsed or CPU time used: 0.150000 s
RESULT: Time elapsed or CPU time used: 0.110000 s
RESULT: Time elapsed or CPU time used: 0.130000 s
TOTAL TIME TO RUN THE CODE 1000 times and repeat the experiment 10 times
=1.250000 s
AVERAGE TIME to run the CODE 1000 times= 0.125000 s
estimated time taken to run the code for one execution( avg time /1000)= 0.125 ms
```

4.3 Binary Detailed Execution Times Result

```
RESULT: Time elapsed or CPU time used: 0.040000 s
RESULT: Time elapsed or CPU time used: 0.020000 s
RESULT: Time elapsed or CPU time used: 0.040000 s
RESULT: Time elapsed or CPU time used: 0.030000 s
RESULT: Time elapsed or CPU time used: 0.040000 s
RESULT: Time elapsed or CPU time used: 0.030000 s
RESULT: Time elapsed or CPU time used: 0.030000 s
RESULT: Time elapsed or CPU time used: 0.030000 s
RESULT: Time elapsed or CPU time used: 0.030000 s
RESULT: Time elapsed or CPU time used: 0.040000 s
TOTAL TIME TO RUN THE CODE 1000 times and repeat the experiment 10 times
=0.330000 s
AVERAGE TIME to run the CODE 1000 times= 0.033000 s
estimated time taken to run the code for one execution( avg time /1000)= 0.033 ms
```

Appendix 5

<u>Steps to run gprof</u>

1.Compile file as follows with -pg:

```
gcc filename.c –pg
```

2.Run the executable

```
./a.out
```

3.Run gprof command

```
gprof  > outputfile
```