

```
/*This Assignment has been done by :  
*Rohit Dhandhanania  
*1 M.Tech(CS)  
*SSSIHL  
*Reg No : 12561  
*/
```

#### Assignment LS1A :

=====ANSWERS=====

#### Exercise 1: Fundamentals

1.How do you run a program in gdb?

Ans : compile the program with -g flag,  
i.e (command-prompt)\$cc -g -o xyz filename.c

2. How do you pass command line arguments to a program when using gdb?

Ans : (gdb-prompt)run [arglist]

3.How do you set a breakpoint in a program?

Ans : (gdb-prompt)break [file:]line  
example :To set the break point at the location 20 in append.c  
(gdb-prompt)b append.c:20  
or,  
(gdb-prompt)break append.c:20

4. How do you set a breakpoint which only occurs when a set of conditions is true (eg when certain variables are a certain value)?

Ans : (gdb-prompt)break ... if expr  
example : (gdb-prompt)break myFunc if var\_name==0

5.How do you execute the next line of C code in the program after a break?

Ans : (gdb-prompt)next  
or,  
(gdb-prompt)step

6.If the next line is a function call, you'll execute the call in one step. How do you execute the C code, line by line, inside the function call?

Ans : (gdb-prompt)step

7.How do you continue running the program after breaking?

Ans : (gdb-prompt)c  
or,  
Ans : (gdb-prompt)continue

8.How can you see the value of a variable (or even an expression) in gdb?

Ans : (gdb-prompt)p variable\_name  
or,

(gdb-prompt)print variable\_name

or,

(gdb-prompt)p /format expression

or,

(gdb-prompt)print /format expression

### Exercise 1: Fundamentals

9.How do you configure gdb so it prints the value of a variable after every step?

Ans : (gdb-prompt)display var\_name

10.How do you print a list of all variables and their values in the current function?

Ans : (gdb-prompt)info locals

or, (gdb-prompt)i lo

11.How do you exit out of gdb?

Ans : (gdb-prompt)quit

or, (gdb-prompt)q

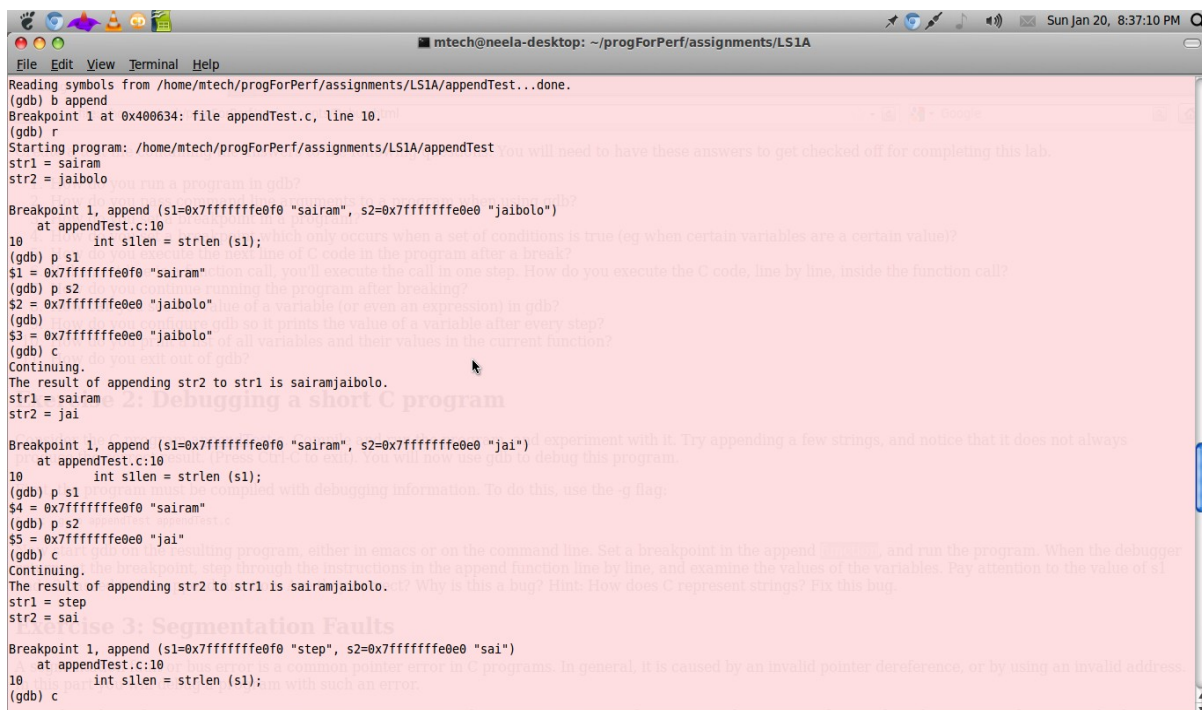
or, (gdb-prompt)Cntrl+d

or, (gdb-prompt)Cntrl+c

### =====Exercise 2: Debugging a short C program=====

Yes,the value of s1 and s2 passed to the append function are correct.


There is a bug because in the append function we are not putting the termination character \0 at the end of the string s1 after appending s2. Hence the previous trace of the string is also being present in area allocated to str1 and we see erroneous results.



```
File Edit View Terminal Help
Reading symbols from /home/mtech/progForPerf/assignments/LS1A/appendTest...done.
(gdb) b append
Breakpoint 1 at 0x400634: file appendTest.c, line 10.
(gdb) r
Starting program: /home/mtech/progForPerf/assignments/LS1A/appendTest
str1 = sairam
str2 = jaibolo
Breakpoint 1, append (s1=0x7fffffff0f0 "sairam", s2=0x7fffffff0f0 "jaibolo")
at appendTest.c:10
10      int silen = strlen (s1);
(gdb) p s1
s1 = 0x7fffffff0f0 "sairam"
(gdb) p s2
s2 = 0x7fffffff0f0 "jaibolo"
(gdb)
s3 = 0x7fffffff0f0 "jaibolo"
(gdb) c
Continuing.
The result of appending str2 to str1 is sairamjaibolo.
str1 = sairam
str2 = jai
Breakpoint 1, append (s1=0x7fffffff0f0 "sairam", s2=0x7fffffff0f0 "jai")
at appendTest.c:10
10      int silen = strlen (s1);
(gdb) p s1
s4 = 0x7fffffff0f0 "sairam"
(gdb) p s2
s5 = 0x7fffffff0f0 "jai"
(gdb) c
Continuing.
The result of appending str2 to str1 is sairamjai.
str1 = step
str2 = sai
Breakpoint 1, append (s1=0x7fffffff0f0 "step", s2=0x7fffffff0f0 "sai")
at appendTest.c:10
10      int silen = strlen (s1);
(gdb) c
```

Fig : Screen shot showing the erroneous behaviour of the append function.

Note : C represents a string as a sequence of characters followed by the termination character `\0` .



```
File Edit View Terminal Help
#include <stdio.h>
#include <string.h>

// Correct Program ....

/*
Return the result of appending the characters in s2 to s1.
Assumption: enough space has been allocated for s1 to store the extra
characters.
*/
char* append (char s1[], char s2[]) {
    int s1len = strlen(s1);
    int s2len = strlen(s2);
    int k;
    for (k=0; k<=s2len; k++) {
        s1[k+s1len] = s2[k];
    }
    return s1;
}

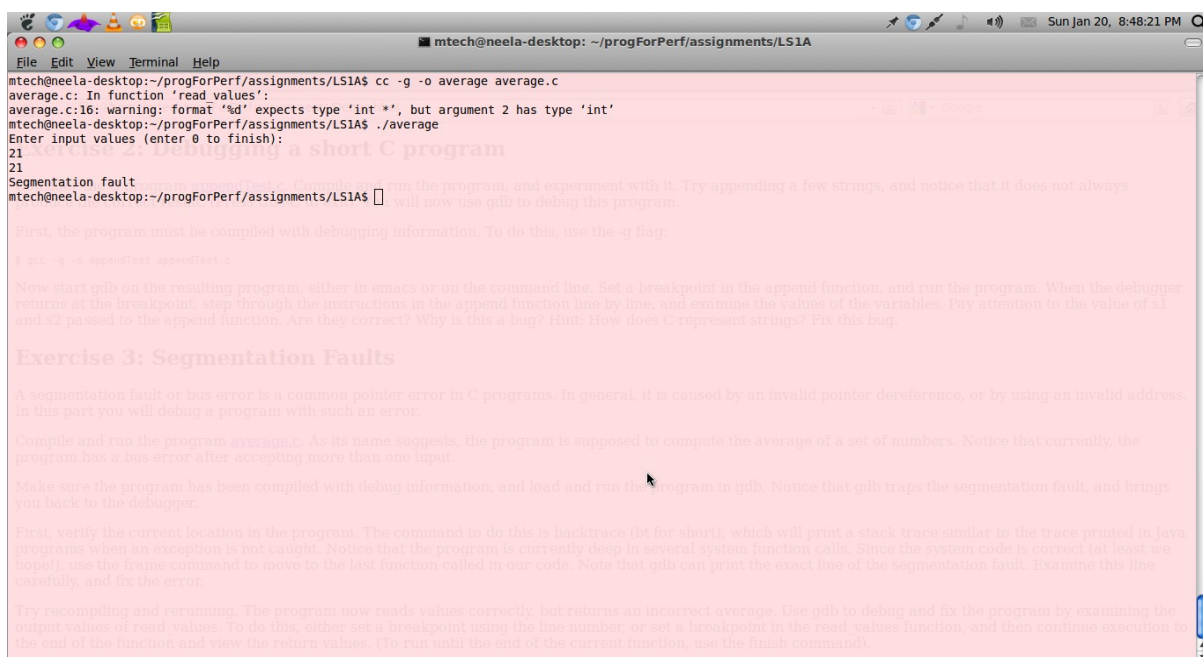
int main () {
    char str1[10];
    char str2[10];
    while (1) {
        printf ("str1 = ");
        if (!gets (str1)) {
            return 0;
        };
        printf ("str2 = ");
        if (!gets (str2)) {
            return 0;
        };
        printf ("The result of appending str2 to str1 is %s.\n",
            append (str1, str2));
    }
    return 0;
}

"appendTest.c" 36L, 785C 1,1 All
```

Fig : Append function after the correction.

Note :The correction is done by also copying the termination character from the s2 to s1.

### =====Exercise 3: Segmentation Faults=====



```
mtech@neela-desktop:~/progForPerf/assignments/LS1A$ cc -g -o average average.c
average.c: In function 'read values':
average.c:16: warning: format '%d' expects type 'int *', but argument 2 has type 'int'
mtech@neela-desktop:~/progForPerf/assignments/LS1A$ ./average
Enter input values (enter 0 to finish):
21
21
Segmentation fault
mtech@neela-desktop:~/progForPerf/assignments/LS1A$
```

Exercise 3: Segmentation Faults

A segmentation fault or bus error is a common pointer error in C programs. In general, it is caused by an invalid pointer dereference, or by using an invalid address. In this part you will debug a program with such an error.

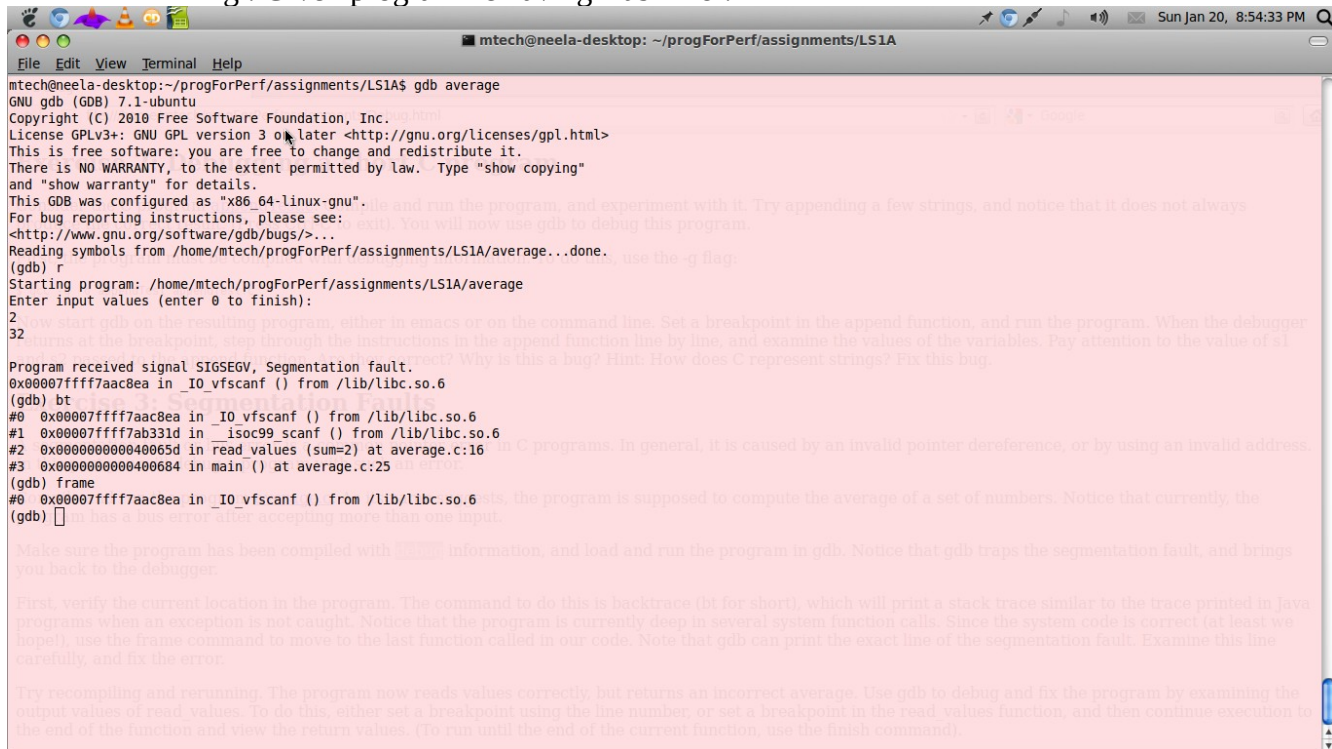
Compile and run the program `average.c`. As its name suggests, the program is supposed to compute the average of a set of numbers. Notice that currently, the program has a bus error after accepting more than one input.

Make sure the program has been compiled with debug information, and load and run the program in gdb. Notice that gdb traps the segmentation fault, and brings you back to the debugger.

First, verify the current location in the program. The command to do this is `backtrace` (or `bt` for short), which will print a stack trace similar to the trace printed in Java programs when an exception is not caught. Notice that the program is currently deep in several system function calls. Since the system code is correct (at least we hope!), use the `frame` command to move to the last function called in our code. Note that gdb can print the exact line of the segmentation fault. Examine this line carefully, and fix the error.

Try recompiling and rerunning. The program now reads values correctly, but returns an incorrect average. Use gdb to debug and fix the program by examining the output values of read values. To do this, either set a breakpoint using the line number, or set a breakpoint in the `read values` function, and then continue execution to the end of the function and view the return values. (To run until the end of the current function, use the `finish` command).

Fig : Given programme having Bus Error.



```
mtech@neela-desktop:~/progForPerf/assignments/LS1A$ gdb average
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/mtech/progForPerf/assignments/LS1A/average...done.
(gdb) r
Starting program: /home/mtech/progForPerf/assignments/LS1A/average
Enter input values (enter 0 to finish):
2
32
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7aac8ea in _IO_vfscanf () from /lib/libc.so.6
(gdb) bt
#0 0x00007ffff7aac8ea in _IO_vfscanf () from /lib/libc.so.6
#1 0x00007ffff7ab331d in __isoc99_scanf () from /lib/libc.so.6
#2 0x00000000040065d in read_values (sum=2) at average.c:16
#3 0x000000000400684 in main () at average.c:25
(gdb) frame
#0 0x00007ffff7aac8ea in _IO_vfscanf () from /lib/libc.so.6
(gdb) 
```

Fig : Use of backtrace and frame command to reach exactly to segmentation fault.



```
#include <stdio.h>
//Still Incorrect
/*
Read a set of values from the user.
Store the sum in the sum variable and return the number of values read.
*/
int read_values(double sum)
{
    int values=0,input=0;
    sum = 0;
    printf("Enter input values (enter 0 to finish):\n");
    scanf("%d",&input);
    while(input != 0) {
        values++;
        sum += input;
        scanf("%d",&input);
    }
    return values;
}

int main()
{
    double sum=0;
    int values;
    values = read_values(sum);
    printf("Average: %g\n",sum/values);
    return 0; as a bus error after accepting more than one input.
}

Make sure the program has been compiled with -g information, and load and run the program in gdb. Notice that gdb traps the segmentation fault, and brings you back to the debugger.

First, verify the current location in the program. The command to do this is backtrace (bt for short), which will print a stack trace similar to the trace printed in Java programs when an exception is not caught. Notice that the program is currently deep in several system function calls. Since the system code is correct (at least we hope!), use the frame command to move to the last function called in our code. Note that gdb can print the exact line of the segmentation fault. Examine this line carefully, and fix the error.

Try recompiling and rerunning. The program now reads values correctly, but returns an incorrect average. Use gdb to debug and fix the program by examining the output values of read values. To do this, either set a breakpoint using the line number, or set a breakpoint in the read_values function, and then continue execution to the end of the function and view the return values. (To run until the end of the current function, use the finish command).
```

Fig : Correct code to overcome segmentation fault.

```
(gdb) p sum
$6 = 0
(gdb) step
11      printf("Enter input values (enter 0 to finish):\n");
(gdb) step
Enter input values (enter 0 to finish):
12      scanf("%d",&input);
(gdb) step
21
13      while(input != 0) {
(gdb) p input
$7 = 21
(gdb) p values
$8 = 0
(gdb) p sum
$9 = 0
(gdb) step
14          values++;
(gdb) p values
$10 = 0
(gdb) step
15          sum += input;
(gdb) p values
$11 = 1
(gdb) step
16          scanf("%d",&input);
(gdb) p sum
$12 = 21
(gdb) p values
$13 = 1
(gdb) p input
$14 = 21
(gdb) step
0
13      while(input != 0) {
(gdb) step
18          return values;
(gdb) p values
$15 = 1
(gdb)
```

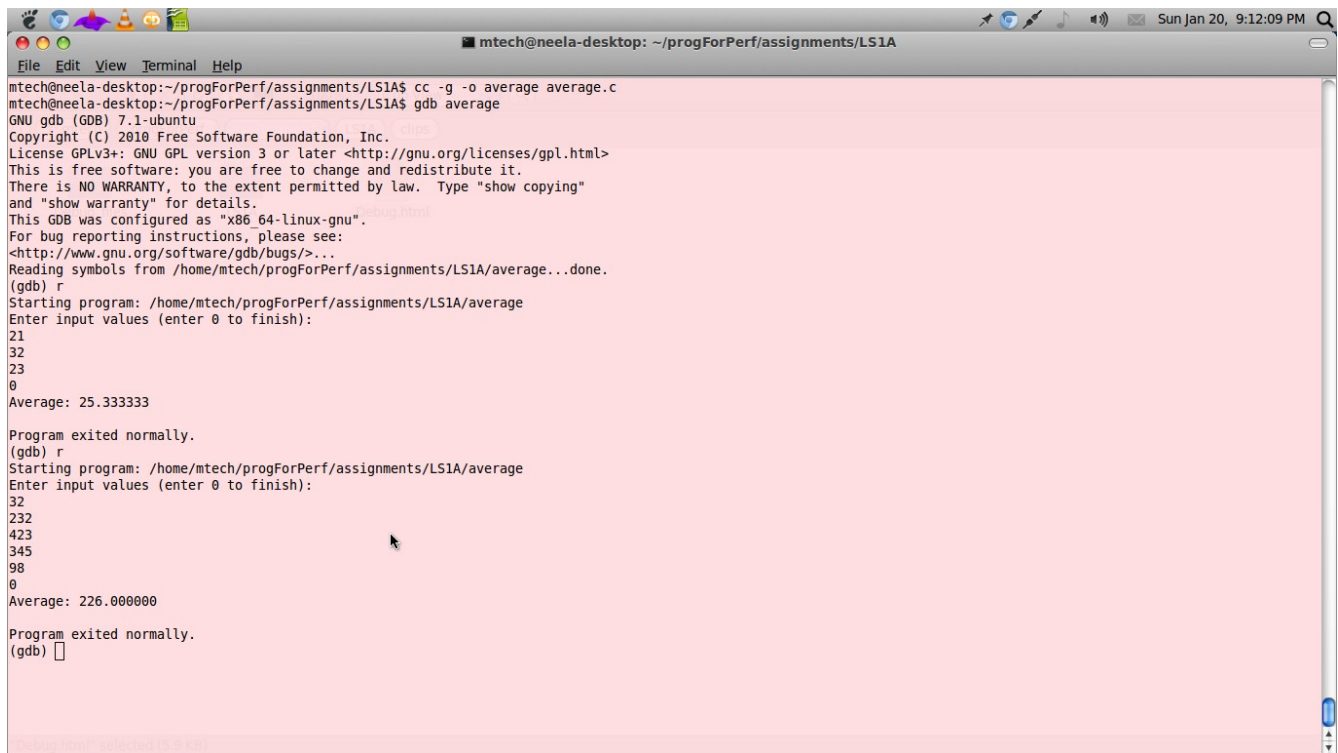
Fig : Running of above code, still giving wrong return values.

```
#include <stdio.h>
//Correct
/*
 * Read a set of values from the user.
 * Store the sum in the sum variable and return the number of values read.
 */
int read_values(double *sum)
{
    int values=0,input=0;
    *sum = 0.0;
    printf("Enter input values (enter 0 to finish):\n");
    scanf("%d",&input);
    while(input != 0) {
        values++;
        *sum += input;
        scanf("%d",&input);
    }
    return values;
}

int main()
{
    double *sum;
    *sum = 0.0;
    int values;
    values = read_values(sum);
    printf("Average: %f\n",*sum/values);
    return 0;
}
```

Fig : Correct Code giving correct results.





```
mtech@neela-desktop: ~/progForPerf/assignments/LS1A
File Edit View Terminal Help
mtech@neela-desktop:~/progForPerf/assignments/LS1A$ cc -g -o average average.c
mtech@neela-desktop:~/progForPerf/assignments/LS1A$ gdb average
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/mtech/progForPerf/assignments/LS1A/average...done.
(gdb) r
Starting program: /home/mtech/progForPerf/assignments/LS1A/average
Enter input values (enter 0 to finish):
21
32
23
0
Average: 25.333333

Program exited normally.
(gdb) r
Starting program: /home/mtech/progForPerf/assignments/LS1A/average
Enter input values (enter 0 to finish):
32
232
423
345
98
0
Average: 226.000000

Program exited normally.
(gdb) □
```

Fig : Execution of the above correct programme.

---

## Assignment LS1B:

### Programming for Performance

#### Lab Session 1: Catch the Bugs.

##### Part A :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
char *strndup(char *, int );
char *strndup(char *src, int max)
{
    char *dest;
    int i;
    if (!src || max <= 0)
        return NULL;
    dest = malloc(max+1);
    for (i=0; i < max && src[i] != 0; i++)
        dest[i] = src[i];
```

```
dest[i] = 0;
return dest;
}
```

Observation : No ,bug found.

=====  
Part B :

/\* Note: For this problem, assume that if the function returns a non- NULL

- pointer to node, then the caller eventually frees node. \*/

```
struct Node {
int data;
struct Node *next;
};
struct List {
struct Node *head;
};
struct Node *push(struct List *, int );
struct Node *push(struct List *list, int data)
{
struct Node *node = (struct Node *)malloc(sizeof(struct Node));
if (!(list && node))
return NULL;
node->data = data;
node->next = list->head;
list->head = node;
return node;
}
```

Observation : Yes , bugs are indeed present.

Bug is memory definitely lost and present in push function call, because we are not freeing up the memory allocated to the node in the push function call.

```
rohit89@login1:~/progForPerf
File Edit View Terminal Help
==23636== For counts of detected and suppressed errors, rerun with: -v
==23636== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
login1$ vi bugs.c
login1$ cc -g -o bugs bugs.c
bugs.c:27: warning: conflicting types for built-in function 'strndup'
login1$ valgrind --tool=memcheck --leak-check=full --show-reachable=yes bugs > log
==24423== Memcheck, a memory error detector
==24423== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==24423== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==24423== Command: bugs
==24423==
3
==24423== Conditional jump or move depends on uninitialised value(s)
==24423==    at 0x4A06D89: strlen (mc_replace_strmem.c:275)
==24423==    by 0x358D446C88: vfprintf (in /lib64/libc-2.5.so)
==24423==    by 0x358D44D549: printf (in /lib64/libc-2.5.so)
==24423==    by 0x4007C0: main (bugs.c:105)
==24423==
==24423== HEAP SUMMARY:
==24423==    in use at exit: 4 bytes in 1 blocks
==24423==    total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==24423==
==24423== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==24423==    at 0x4A05E1C: malloc (vg_replace_malloc.c:195)
==24423==    by 0x4005CB: strndup (bugs.c:34)
==24423==    by 0x4007AE: main (bugs.c:105)
==24423==
==24423== LEAK SUMMARY:
==24423==    definitely lost: 4 bytes in 1 blocks
==24423==    indirectly lost: 0 bytes in 0 blocks
==24423==    possibly lost: 0 bytes in 0 blocks
==24423==    still reachable: 0 bytes in 0 blocks
==24423==    suppressed: 0 bytes in 0 blocks
==24423==
==24423== For counts of detected and suppressed errors, rerun with: -v
==24423== Use --track-origins=yes to see where uninitialised values come from
==24423== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
login1$ view log
login1$
```

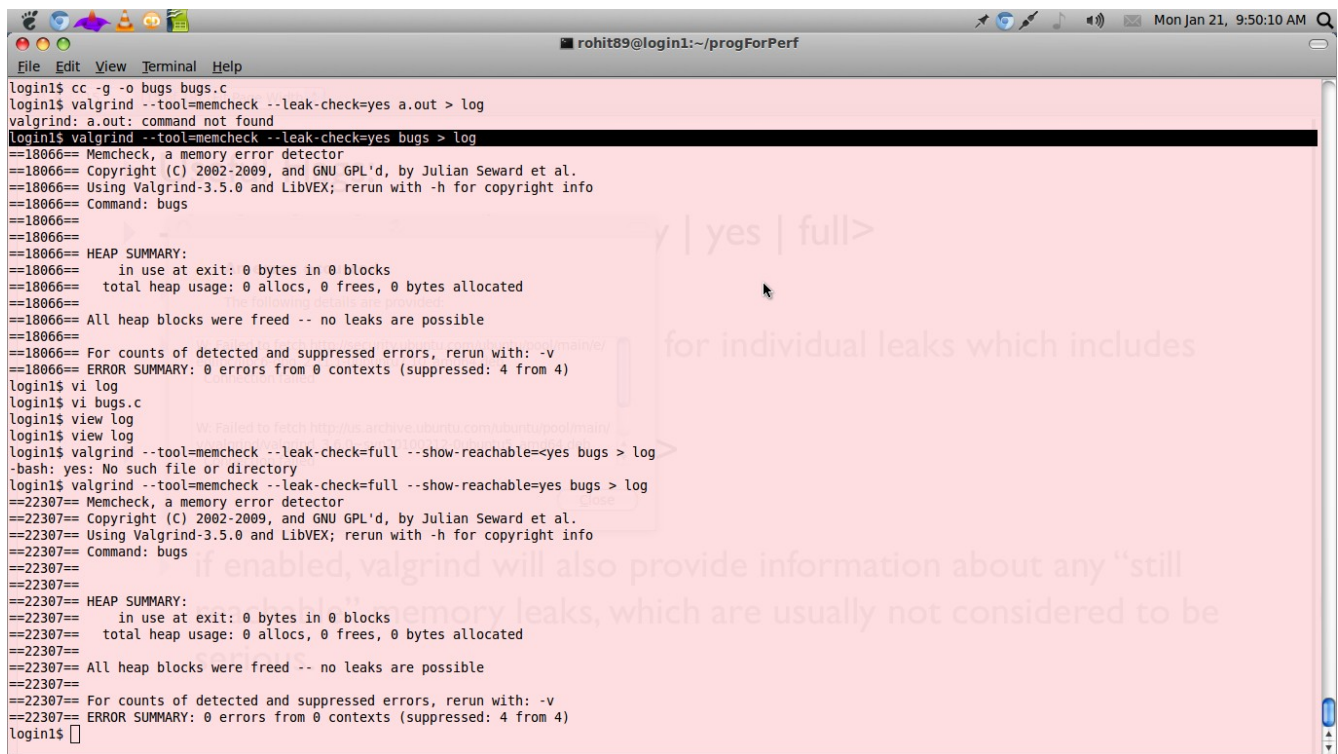
Fig : Screen Shot Showing the existence of bugs in PartB.

Part C :

```
/* print_shortest - prints the shortest of two strings */
char *shortest(char *, char *);
char *shortest(char *str1, char *str2)
{
    char *equal = "equal";
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    if (len1 == len2)
        return equal;
    else
        return (len1 < len2 ? str1 : str2);
}
void print_shortest(char *, char *);
void print_shortest(char *str1, char *str2)
{
    printf("The shortest string is %s \n",shortest(str1, str2));
}
```

Observation : NO, bug is present as reported by valgrind. See the clips down.





The image shows a terminal window titled 'rohit89@login1:~/progForPerf'. The user has compiled a program 'bugs.c' and run it with Valgrind. The output shows that the program executed successfully with no memory errors or leaks detected. The terminal text is as follows:

```
login1$ cc -g -o bugs bugs.c
login1$ valgrind --tool=memcheck --leak-check=yes a.out > log
valgrind: a.out: command not found
login1$ valgrind --tool=memcheck --leak-check=yes bugs > log
==18066== Memcheck, a memory error detector
==18066== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==18066== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==18066== Command: bugs
==18066==
==18066== HEAP SUMMARY:
==18066==   in use at exit: 0 bytes in 0 blocks
==18066== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==18066== All heap blocks were freed -- no leaks are possible
==18066== For counts of detected and suppressed errors, rerun with: -v
==18066== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
login1$ vi log
login1$ vi bugs.c
login1$ view log
login1$ view log
login1$ valgrind --tool=memcheck --leak-check=full --show-reachable=yes bugs > log
-bash: yes: No such file or directory
login1$ valgrind --tool=memcheck --leak-check=full --show-reachable=yes bugs > log
==22307== Memcheck, a memory error detector
==22307== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==22307== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==22307== Command: bugs
==22307==
==22307== HEAP SUMMARY:
==22307==   in use at exit: 0 bytes in 0 blocks
==22307== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==22307== All heap blocks were freed -- no leaks are possible
==22307== For counts of detected and suppressed errors, rerun with: -v
==22307== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
login1$
```

Fig : Screen Shot Showing the existence of no bugs in PartC.