/*This Assignment has been done by :
*Rohit Dhandhania
*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 seee eroneous results.

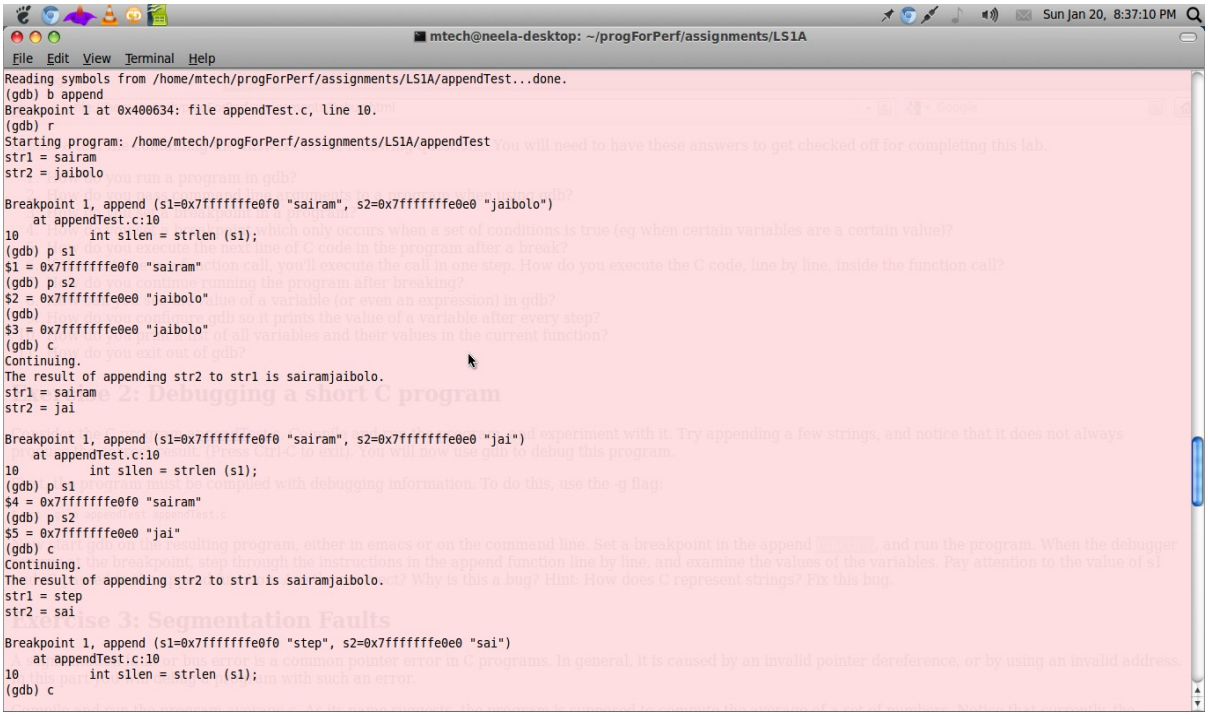



Fig : Screen shot showing the erroneous behaviour of the append function.

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

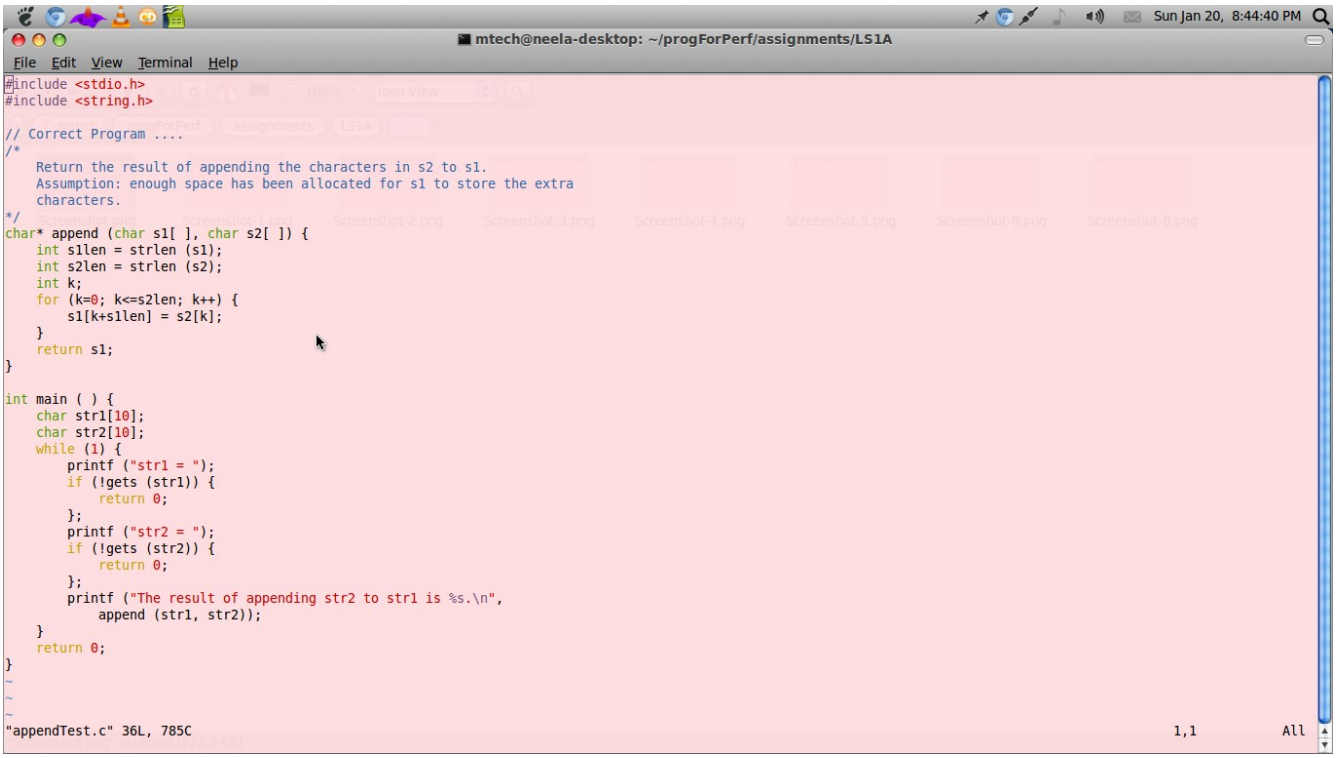



Fig : Append function after the correction.

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

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

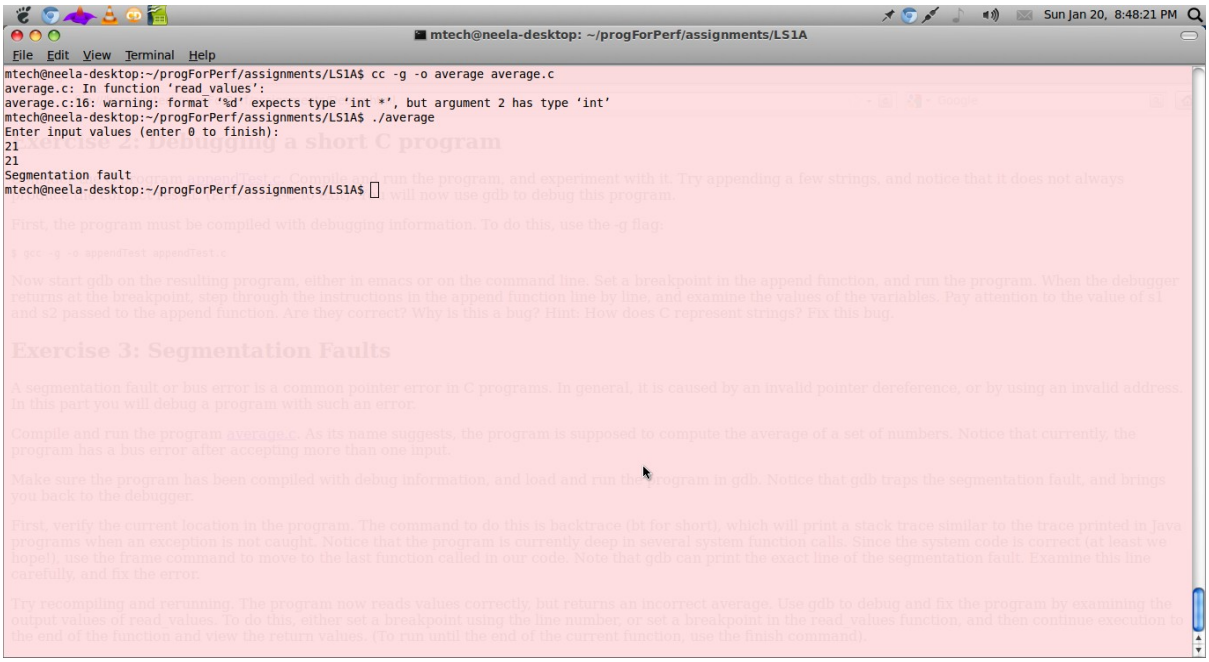


Fig : Given programme having Bus Error.

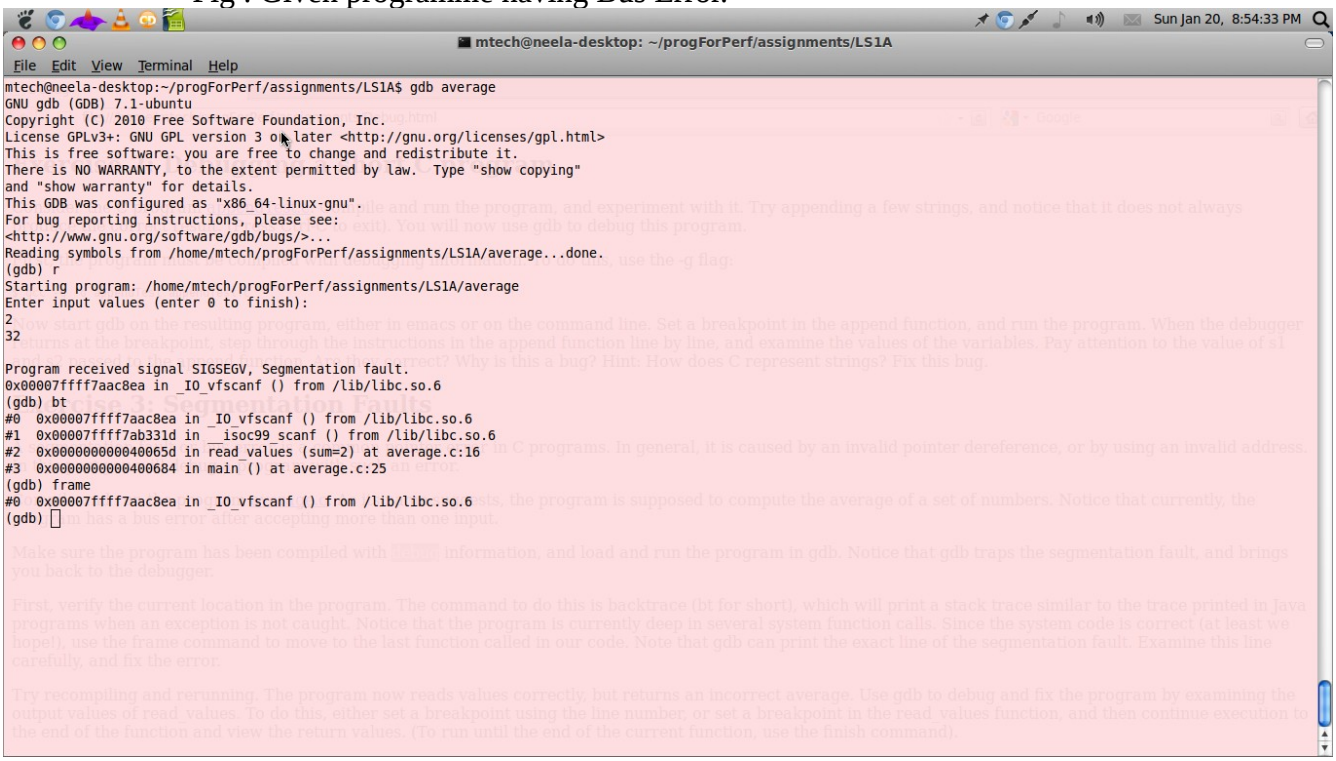



Fig : Use of backtrace and frame command to reach exatly to segmentation fault.

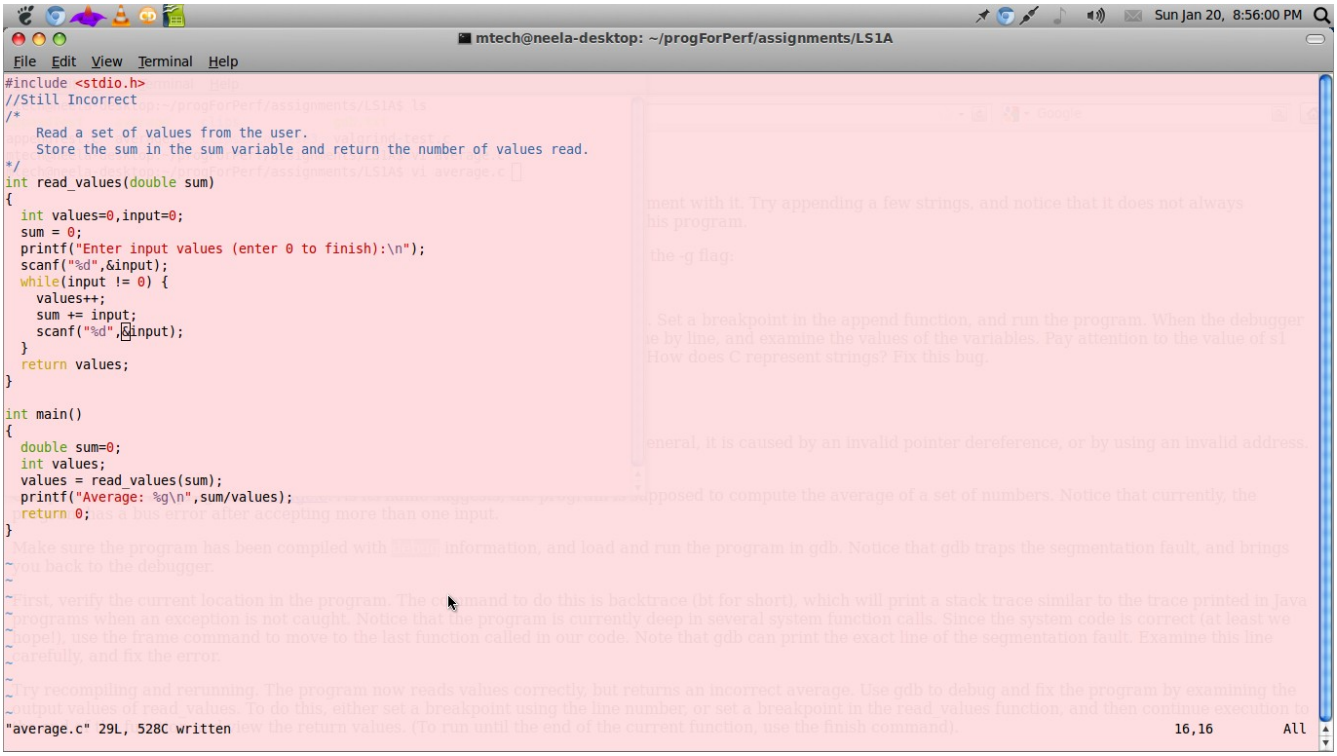



Fig : Correct code to overcome segmentation fault.

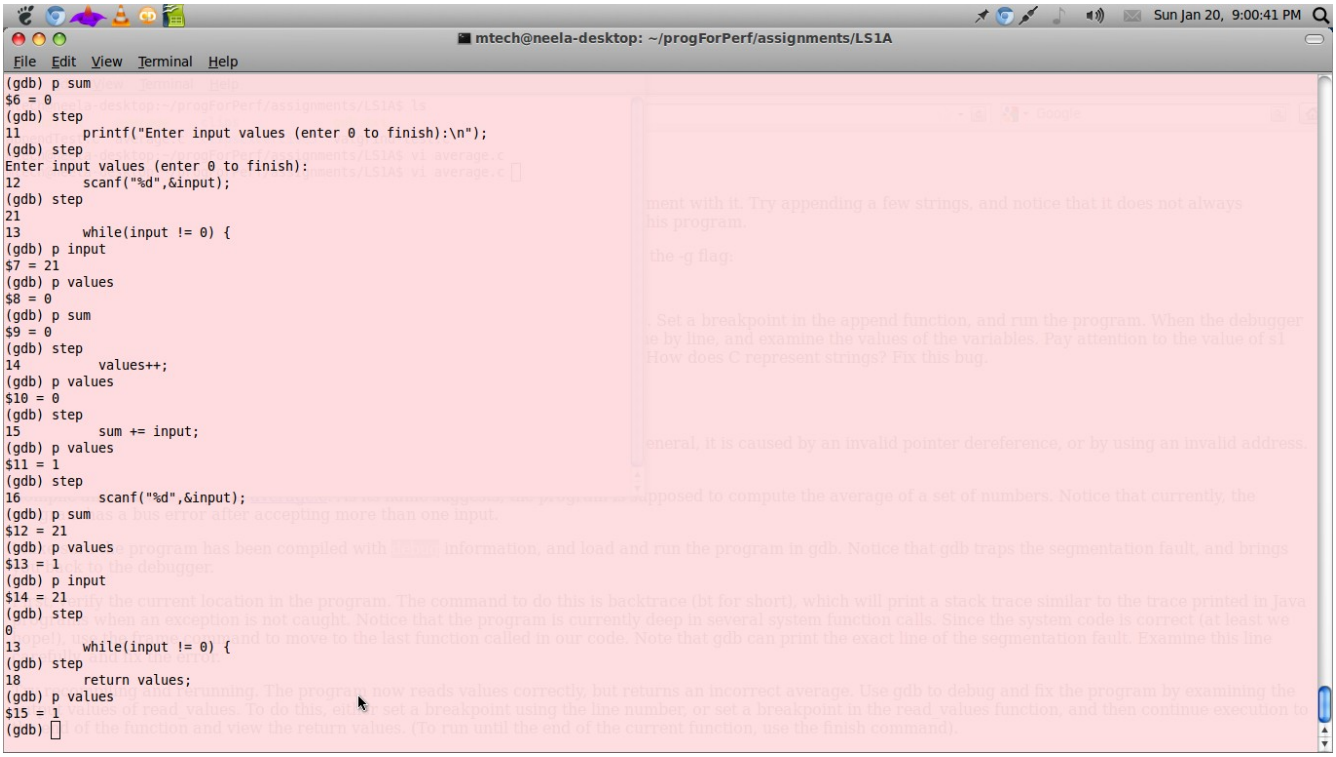



Fig : Running of above code, still giving wrong return values.


Fig : Correct Code giving correct results.

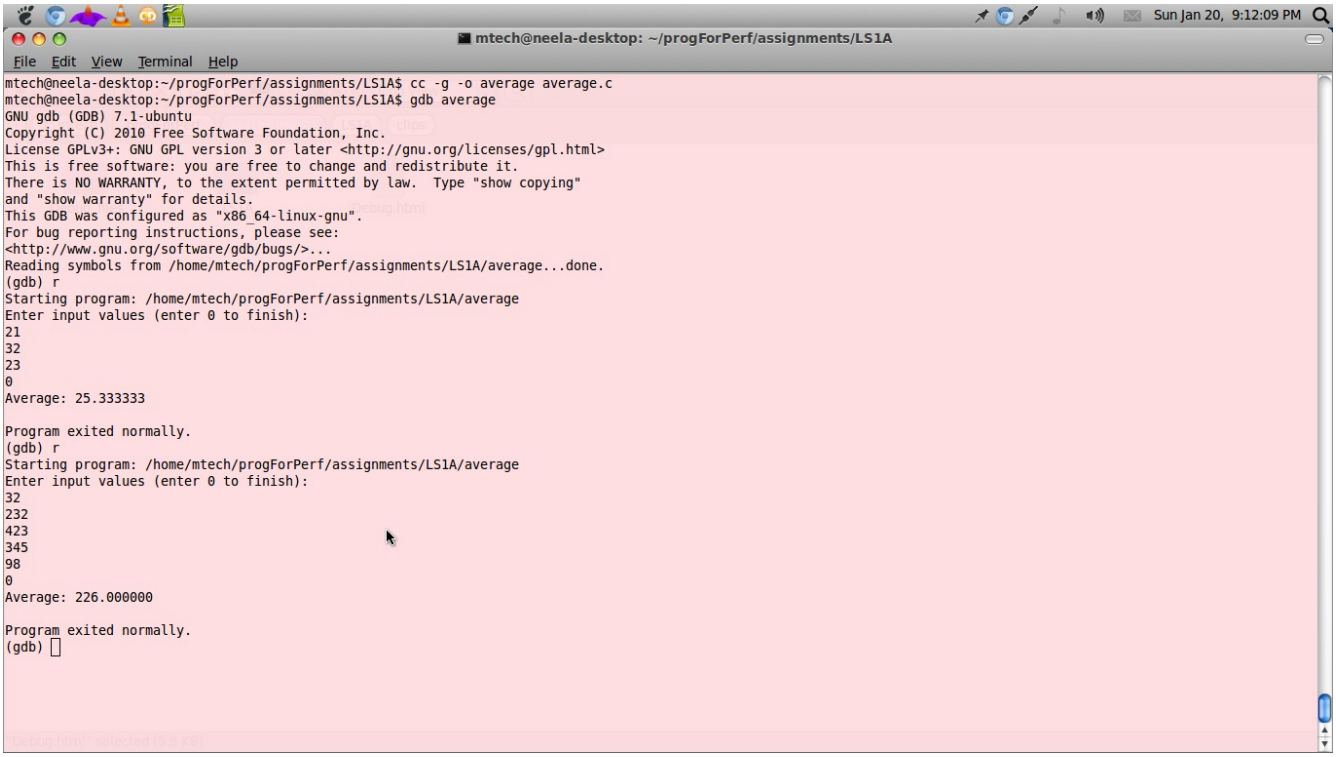



Fig : Execution of the above correct programme.

---

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, asssume 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.

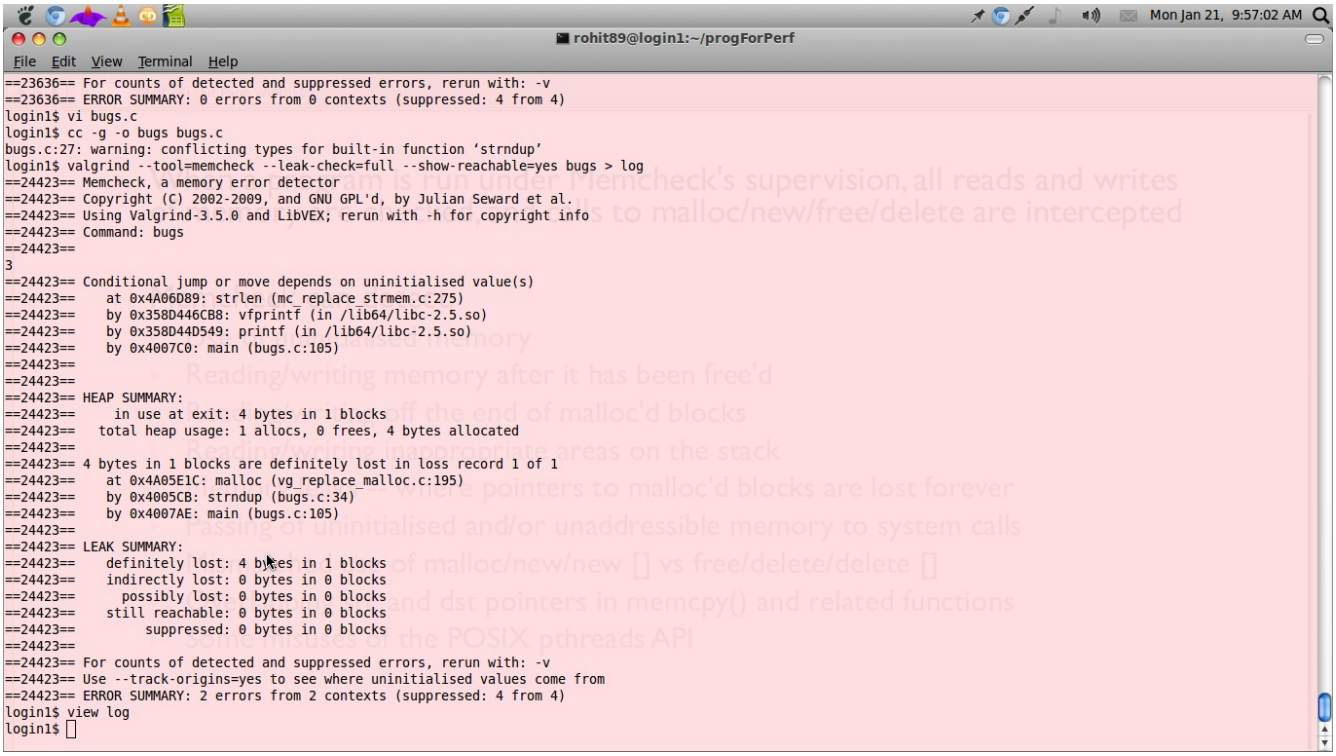



Fig : Screen Shot Showing the existence of bugs in PartB.

==========

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.

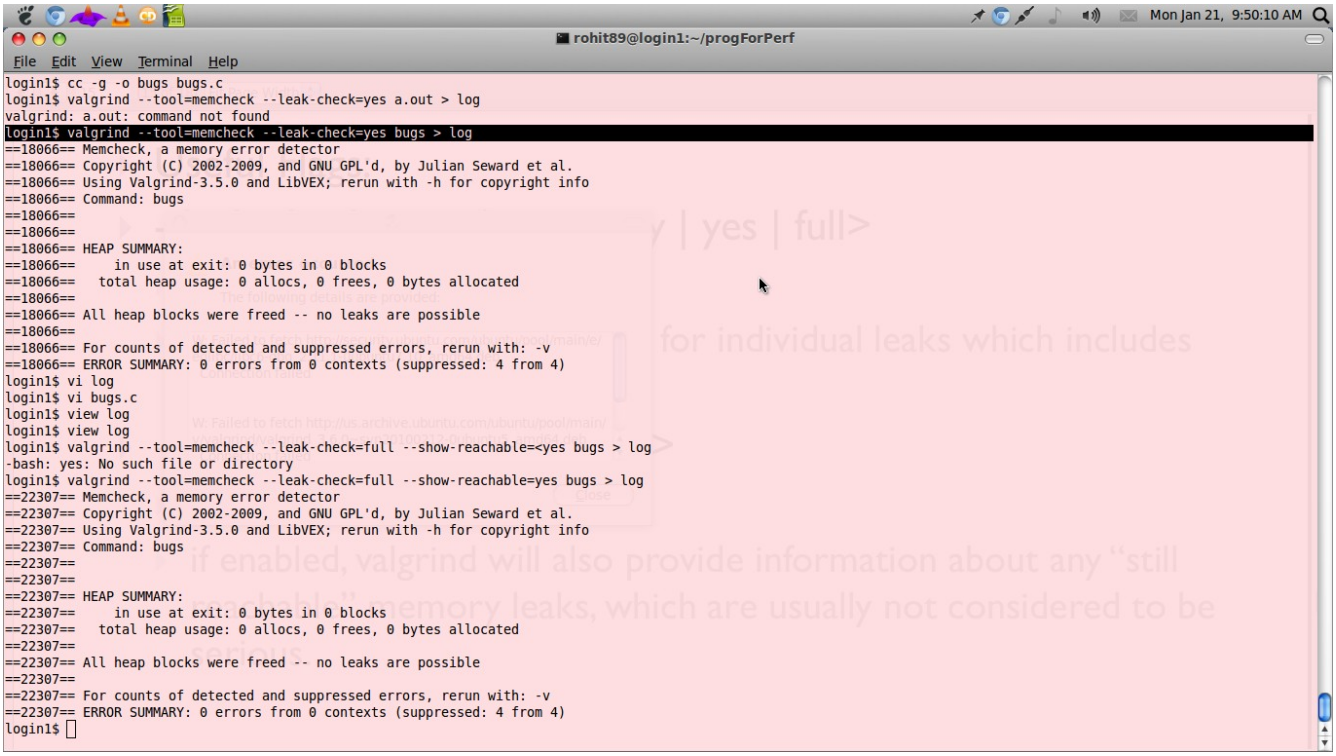
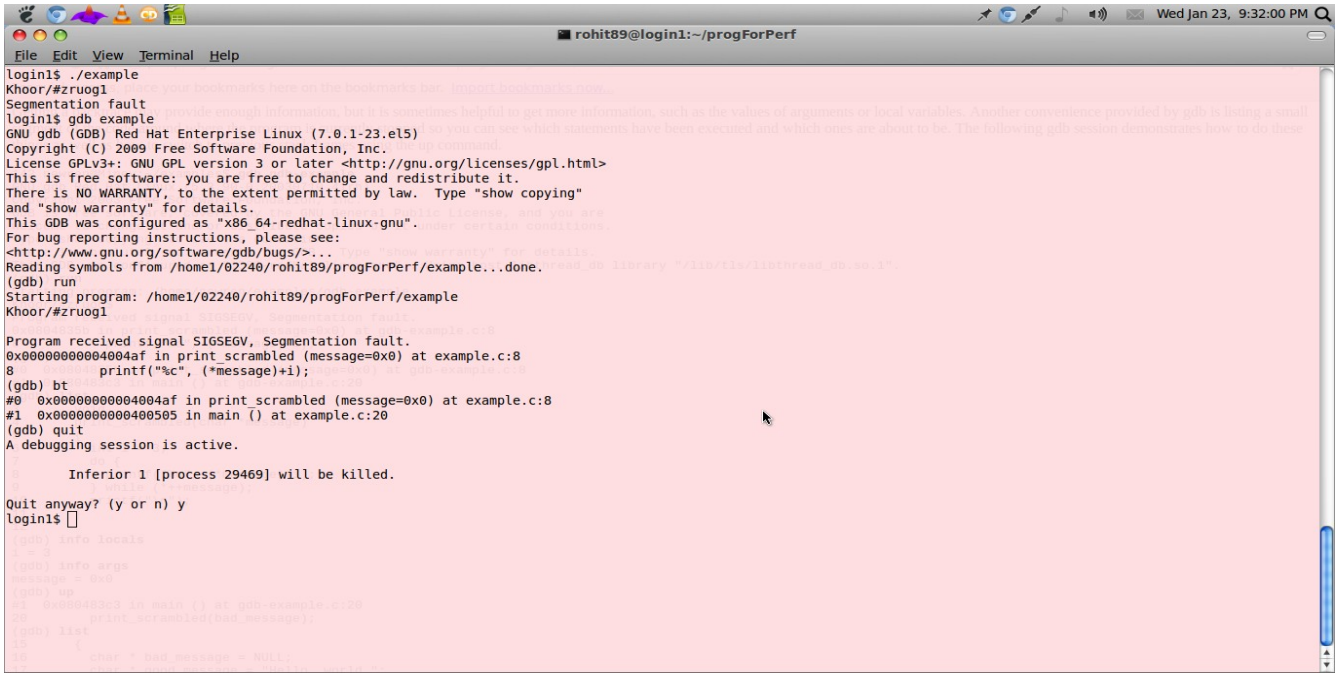



Fig : Screen Shot Showing the existence of no bugs in PartC.

---

Assignment LS1C :

Observation : The function print_scrambled when called on bad message variable as argument,was trying to add 3 to a NULL pointer and then trying to print it, which is wrong.Hence we got segmentation fault.



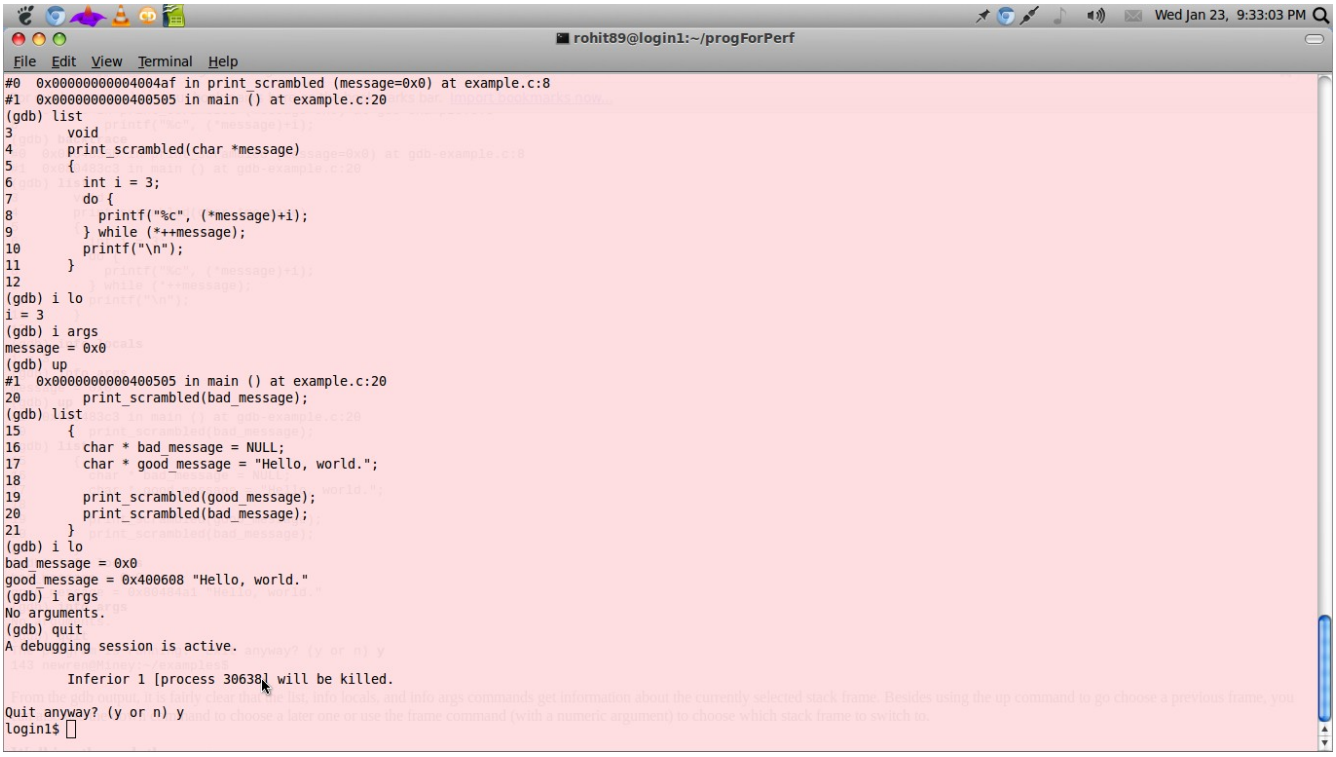



Fig : Screen shot of the steps as asked in LS1C assignment.

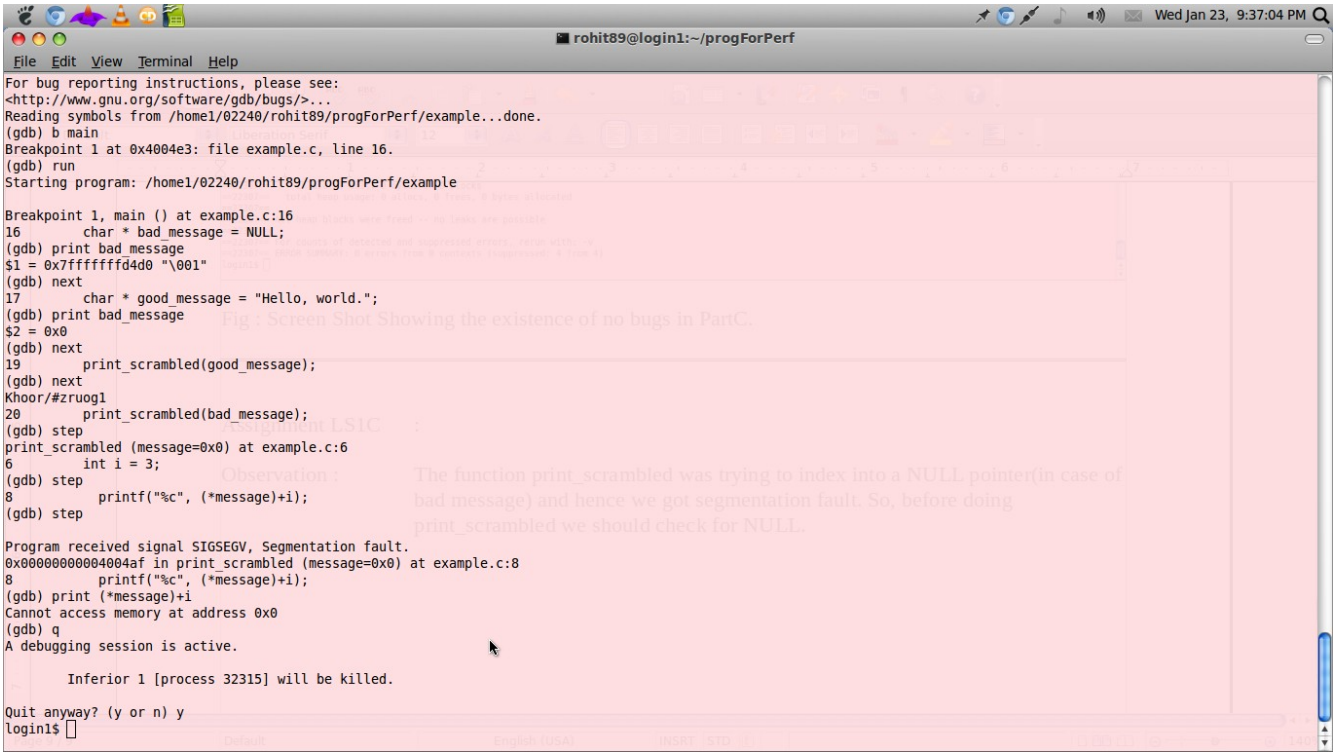



Fig : Screen shot of the steps as asked in LS1C assignment.

Soluiton : Thus we should check for null values before the do-while loop in print_scrambled function

….....Jai Sai Ram ….....