Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Principles of Embedded Software

Project 1 Report

By
Mohit Rane
&
Suman Hosmane

Under the Guidance of
Prof. Kevin Gross

Department of Electrical, Computer and Energy Engineering
University of Colorado Boulder
Boulder, CO - 80309
October 7th, 2018

Index

Chapter 1

Introduction

This project demonstrates how to use Makefile to consolidate various .c, .o and .h file into a single project or a single main.c file. We have various functionalities incorporated in the project, listed below:

-help

help command invokes the help function which tells the user as to how to use the different functions (just like the general Linux help command) and gives the syntax and purpose of the functions defined.

-allo

allocates a block of memory as required by a user. Displays the base address from which the allocation starts from.

-free

releases the block of allocated memory in the allocate function.

-write

asks for an offset from user and feeds the data provided into that offset location.

-display

reads back data from the memory location specified by the user.

-random

generates up to 5 random numbers from a seed value provided by the user and stores it in memory spaces starting from an offset value.

-verify

verifies that the previous random pattern with the new set of values from user and if a match occurs, pattern is verified, else displays address of mismatch and values expected and calculated.

-exit

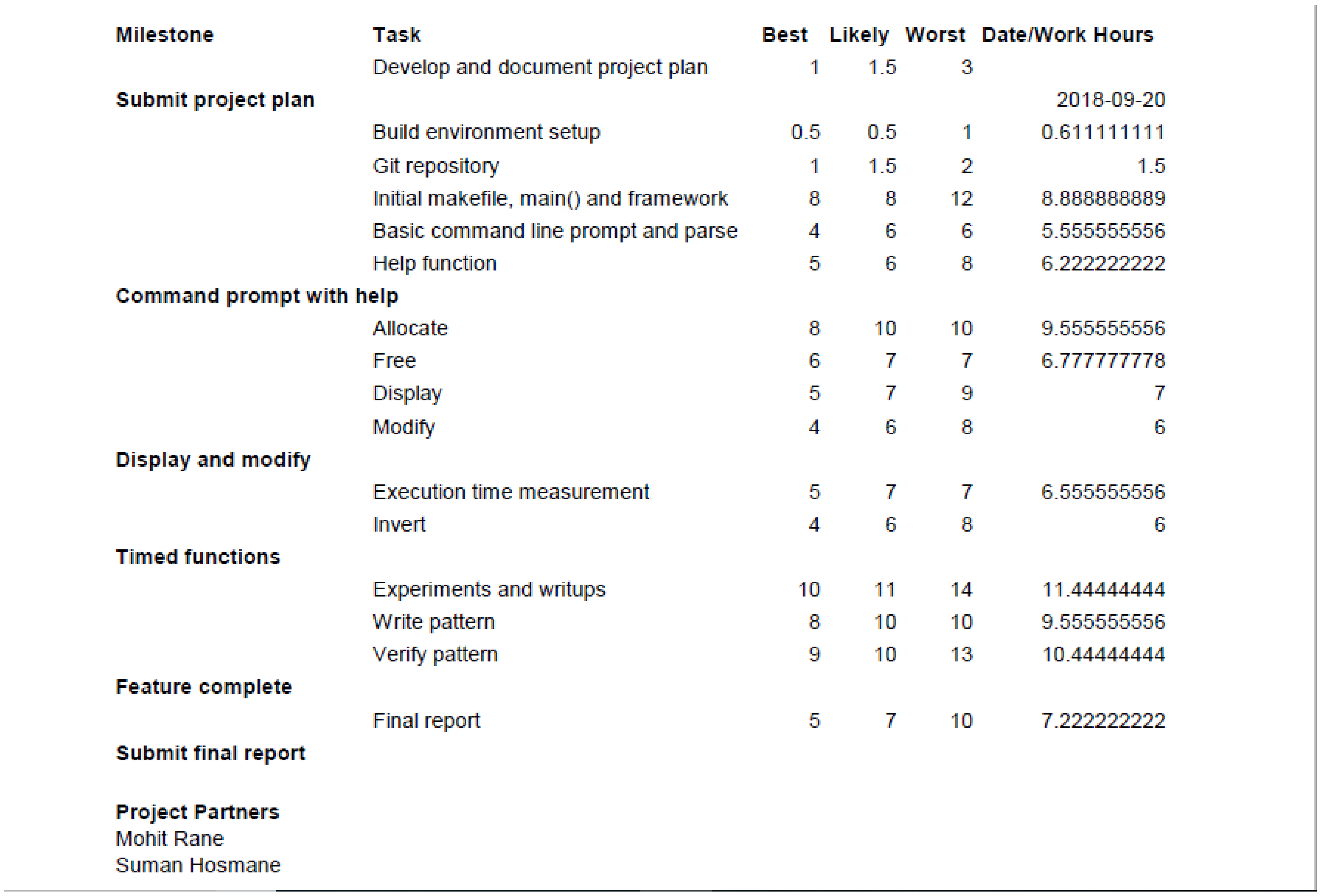
the entire program is in a loop and the exit command can be invoked to terminate the program.

While making the project we have taken into consideration some of the boundary conditions that could occur such as performing free or any other memory requiring operation without allocating memory. Or an invalid user input which will show an invalid flag instead of breaking out of the program.

Basic testing is done via a **stdin** test file with all the functions to be tested along with the required parameters for each function. The compiler automatically feeds the parameters after each function invoked in the stdin file.

Chapter 2

Project Plan Estimate



Milestone	Task	Best	Likely	Worst	Date/Work Hours
	Develop and document project plan	1	1.5	3	
Submit project plan					2018-09-20
	Build environment setup	0.5	0.5	1	0.611111111
	Git repository	1	1.5	2	1.5
	Initial makefile, main() and framework	8	8	12	8.888888889
	Basic command line prompt and parse	4	6	6	5.555555556
	Help function	5	6	8	6.222222222
Command prompt with help					
	Allocate	8	10	10	9.555555556
	Free	6	7	7	6.777777778
	Display	5	7	9	7
	Modify	4	6	8	6
Display and modify					
	Execution time measurement	5	7	7	6.555555556
	Invert	4	6	8	6
Timed functions					
	Experiments and writups	10	11	14	11.44444444
	Write pattern	8	10	10	9.555555556
	Verify pattern	9	10	13	10.44444444
Feature complete					
	Final report	5	7	10	7.222222222
Submit final report					

Project Partners
Mohit Rane
Suman Hosmane

**This was the estimate we made earlier to do each part of the project. After completing the project, we found out that it took us approximately 55 hours combined which is much lesser than what we originally thought.

Chapter 3

Documentation for Syntax

The available functions are:

To go to help section -------------- <help>

To allocate memory --------------- <allo>

To write to memory --------------- <write>

To display from memory --------- <display>

To invert from memory ----------- <invert>

To generate random numbers ---- <random>

To verify pattern ------------------- <verify>

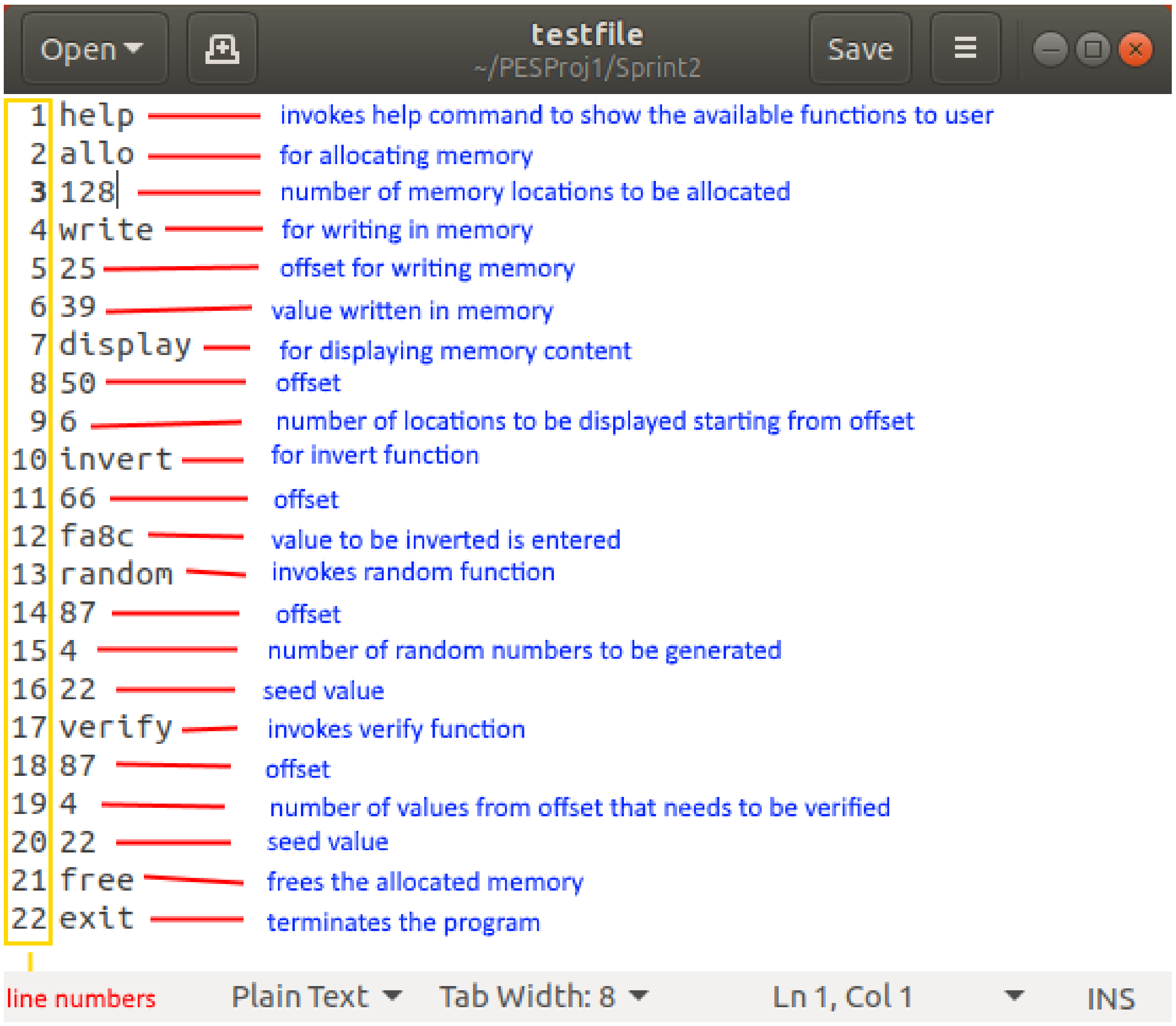
To exit program -------------------- <exit>

→ For the parameters that are required for the functions, the user is prompted to enter them after he activates that function by typing the function name in CLI.

Chapter 4

Test Case input file

For the testing phase, a simple stdin file was passed to the executable file with functions to be invoked along with parameters to run each command.



help	invokes help command to show the available functions to user
allo	for allocating memory
128	number of memory locations to be allocated
write	for writing in memory
25	offset for writing memory
39	value written in memory
display	for displaying memory content
50	offset
6	number of locations to be displayed starting from offset
invert	for invert function
66	offset
fa8c	value to be inverted is entered
random	invokes random function
87	offset
4	number of random numbers to be generated
22	seed value
verify	invokes verify function
87	offset
4	number of values from offset that needs to be verified
22	seed value
free	frees the allocated memory
exit	terminates the program

Chapter 5

Answers to Report questions

1. **How well did your effort estimates match with actual effort? Provide examples of both inaccurate and accurate estimates and discuss any contributing factors to the success or failings of your effort estimates**.

   In our experience of the project, many things lagged that predicted. We ran into various unexpected hurdles while trying to resolve boundary conditions and converting our logic into a code. We also faced PC (Linux VM) failure occasionally. Though we could make up for it by reducing time spent on easier functions such as help, allocate and exit.

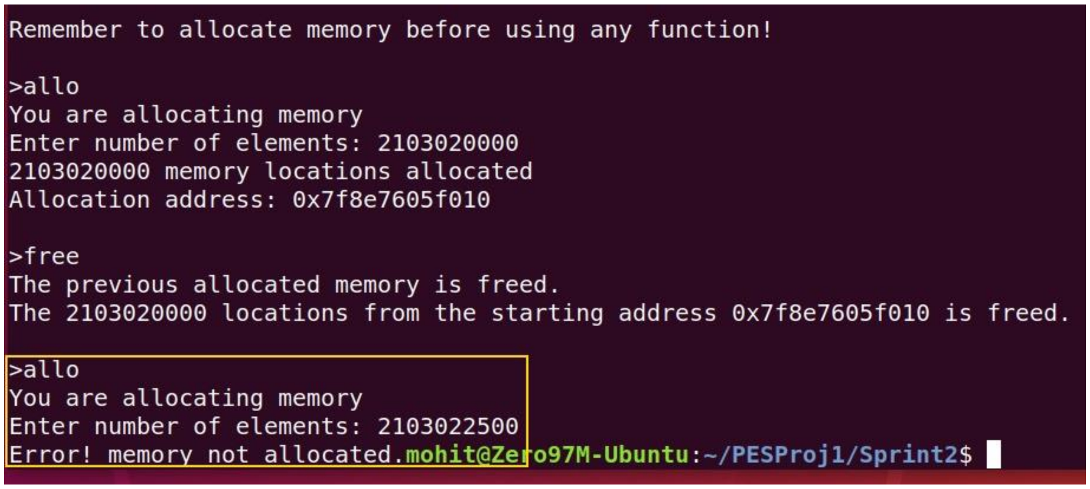
2. **Were there any tasks that you forgot to include in your project plan?**

   We planned the project dedicating most time towards code, boundary conditions and reducing lines of code. However, we forgot to include the testing phase portion in project plan which is as important part when running the code in real time. Extensive testing is not done on the code, however most of the boundary conditions are taken care of in the development phase.

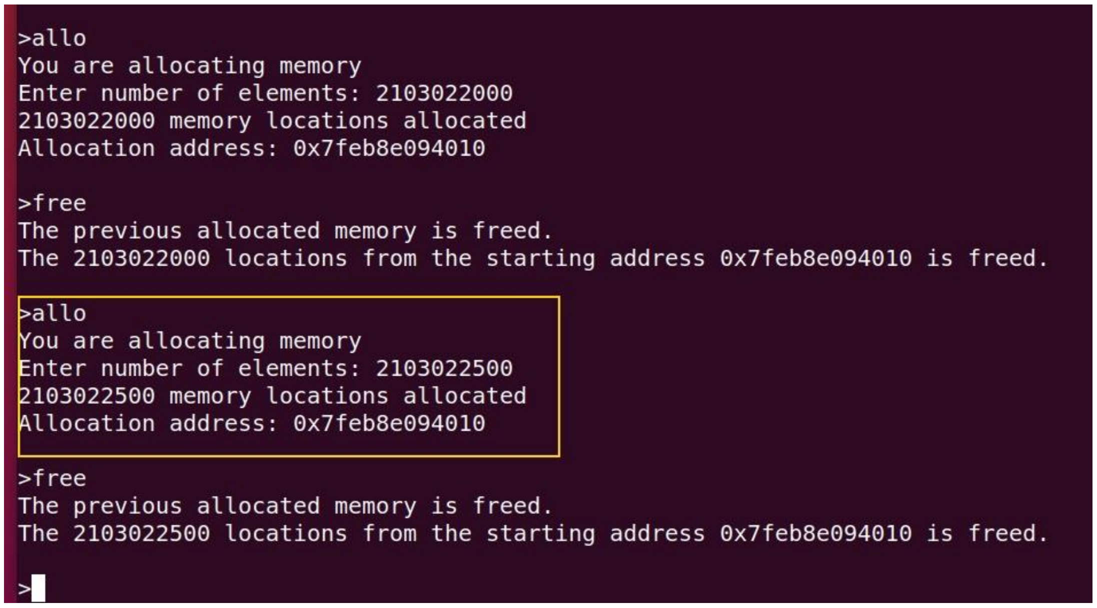
3. **What is the largest block of memory you can allocate? Discuss.**

   We could approximately allocate a block of up to 2000000000 ~(7.45GB). But this is not a definitive number since the allocation limit depends upon the base address (start of allocation).

   We can see this clearly in the following images that we were able to allocate 2103022500 memory locations in one iteration however the next time the program gave error for allocating the same memory locations.



```
Remember to allocate memory before using any function!

>allo
You are allocating memory
Enter number of elements: 2103020000
2103020000 memory locations allocated
Allocation address: 0x7f8e7605f010

>free
The previous allocated memory is freed.
The 2103020000 locations from the starting address 0x7f8e7605f010 is freed.

>allo
You are allocating memory
Enter number of elements: 2103022500
Error! memory not allocated.mohit@Zero97M-Ubuntu:~/PESProj1/Sprint2$
```


```
>allo
You are allocating memory
Enter number of elements: 2103022000
2103022000 memory locations allocated
Allocation address: 0x7feb8e094010

>free
The previous allocated memory is freed.
The 2103022000 locations from the starting address 0x7feb8e094010 is freed.

>allo
You are allocating memory
Enter number of elements: 2103022500
2103022500 memory locations allocated
Allocation address: 0x7feb8e094010

>free
The previous allocated memory is freed.
The 2103022500 locations from the starting address 0x7feb8e094010 is freed.

>
```

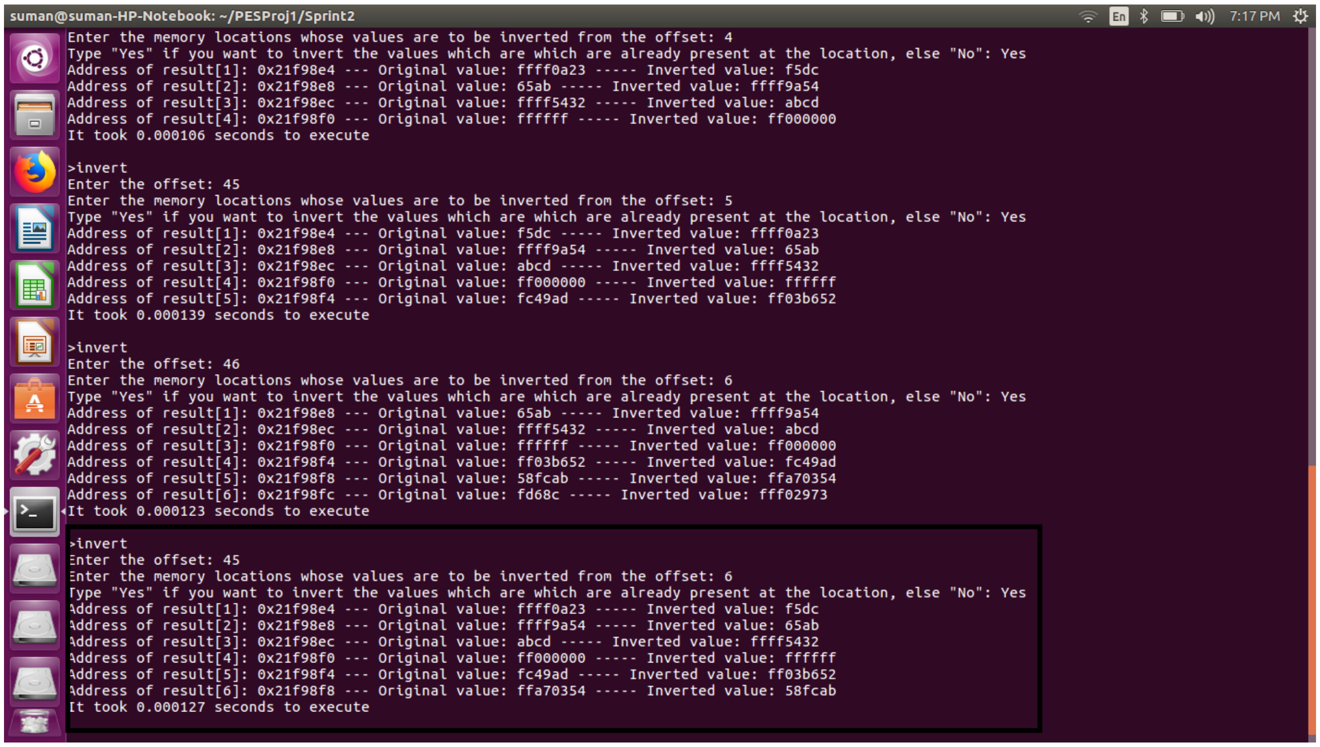
4. **What happens if you try to read outside of the allocated block?**

   The boundary condition we defined took care of this problem. If the user tries to read any data outside the size of his allocated block, the system asked the user to ‘Try again, since he/she is accessing data outside allocated block.’ The same rule is taken in for all functions that need memory access.

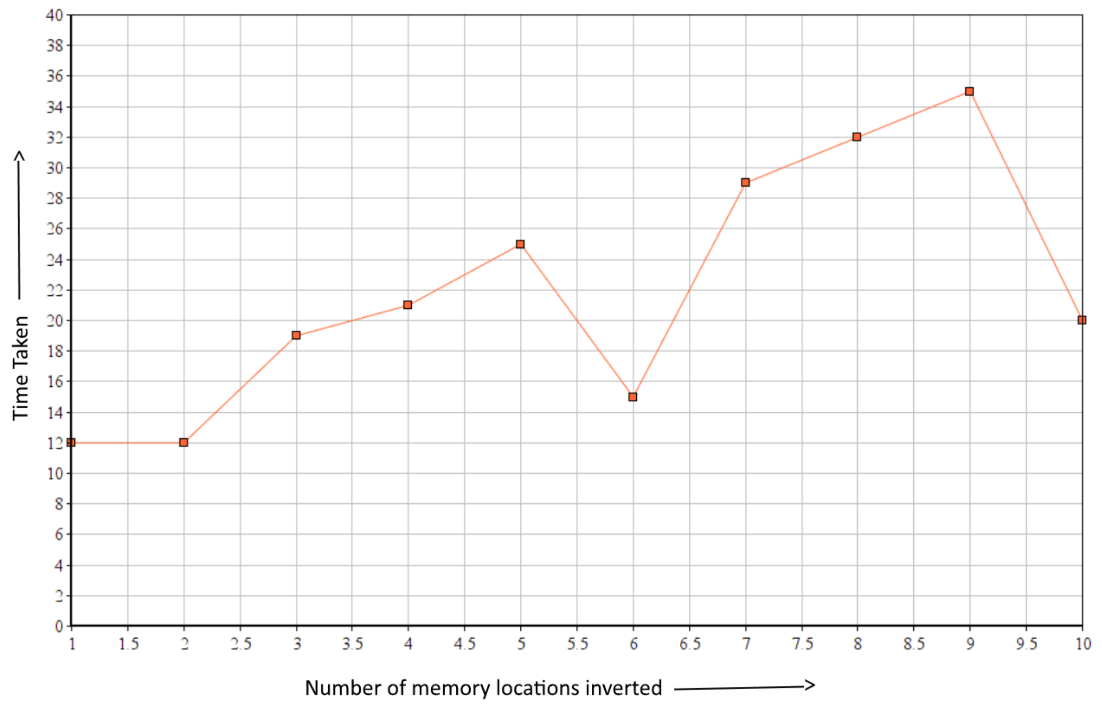
5. **What happens if you try to write outside of the allocated block?**

   Same as read. The code shows a warning message asking the user to write into memory he/she has allocated and not outside. Condition is checked if the offset value is within the memory space allocated by the user.

6. **Analyse the time it takes to invert memory for different memory block sizes. Is there a linear relationship between time and size?**



```
Enter the memory locations whose values are to be inverted from the offset: 4
Type "Yes" if you want to invert the values which are which are already present at the location, else "No": Yes
Address of result[1]: 0x21f98e4 --- Original value: ffff0a23 ----- Inverted value: f5dc
Address of result[2]: 0x21f98e8 --- Original value: 65ab ----- Inverted value: ffff9a54
Address of result[3]: 0x21f98ec --- Original value: ffff5432 ----- Inverted value: abcd
Address of result[4]: 0x21f98f0 --- Original value: ffffff ----- Inverted value: ff000000
It took 0.000106 seconds to execute

>invert
Enter the offset: 45
Enter the memory locations whose values are to be inverted from the offset: 5
Type "Yes" if you want to invert the values which are which are already present at the location, else "No": Yes
Address of result[1]: 0x21f98e4 --- Original value: f5dc ----- Inverted value: ffff0a23
Address of result[2]: 0x21f98e8 --- Original value: ffff9a54 ----- Inverted value: 65ab
Address of result[3]: 0x21f98ec --- Original value: abcd ----- Inverted value: ffff5432
Address of result[4]: 0x21f98f0 --- Original value: ff000000 ----- Inverted value: ffffff
Address of result[5]: 0x21f98f4 --- Original value: fc49ad ----- Inverted value: ff03b652
It took 0.000139 seconds to execute

>invert
Enter the offset: 46
Enter the memory locations whose values are to be inverted from the offset: 6
Type "Yes" if you want to invert the values which are which are already present at the location, else "No": Yes
Address of result[1]: 0x21f98e8 --- Original value: 65ab ----- Inverted value: ffff9a54
Address of result[2]: 0x21f98ec --- Original value: ffff5432 ----- Inverted value: abcd
Address of result[3]: 0x21f98f0 --- Original value: ffffff ----- Inverted value: ff000000
Address of result[4]: 0x21f98f4 --- Original value: ff03b652 ----- Inverted value: fc49ad
Address of result[5]: 0x21f98f8 --- Original value: 58fcab ----- Inverted value: ffa70354
Address of result[6]: 0x21f98fc --- Original value: fd68c ----- Inverted value: fff02973
It took 0.000123 seconds to execute

>invert
Enter the offset: 45
Enter the memory locations whose values are to be inverted from the offset: 6
Type "Yes" if you want to invert the values which are which are already present at the location, else "No": Yes
Address of result[1]: 0x21f98e4 --- Original value: ffff0a23 ----- Inverted value: f5dc
Address of result[2]: 0x21f98e8 --- Original value: ffff9a54 ----- Inverted value: 65ab
Address of result[3]: 0x21f98ec --- Original value: abcd ----- Inverted value: ffff5432
Address of result[4]: 0x21f98f0 --- Original value: ff000000 ----- Inverted value: ffffff
Address of result[5]: 0x21f98f4 --- Original value: fc49ad ----- Inverted value: ff03b652
Address of result[6]: 0x21f98f8 --- Original value: ffa70354 ----- Inverted value: 58fcab
It took 0.000127 seconds to execute
```



This graph shows the time taken to invert 1 to 10 values in each iteration. We can see that the graph is linear most of the time but there are few exceptions. The unusual skewing of the graph could not be explained/understood by us very clearly since we could not come up with any logical explanation for sudden drop in execution time for a particular number of operands like 6 and 10 in our case.

7. **What are the limitations of your time measurements?**

   We did time measurements using the system clock function. Since the clock frequency is hardware dependent we cannot get the same time of execution across different devices. Additionally, it also considers time taken to execute the clock function itself. This can add to the output value of run time.

8. **What improvements you can make to the invert function, your program or your build, to make it run faster? How much improvement did you achieve?**

   We used a XOR function to make the inversion. Probably a NOT operator will reduce execution time and lines of code. In XOR we had 2 operands (user input and ffffffff) which requires 2 machine cycles to fetch and decode, while a NOT has only one operand.

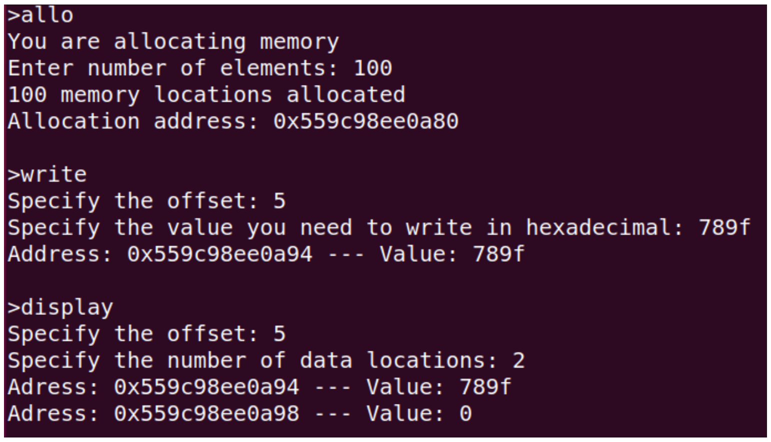
9. **What algorithm did you choose for your pattern generator? How does it generate its pattern? What are its strengths and weaknesses?**

   We implemented a random number using the time() function which gives the current time as parameter. A set of random mathematical operators were used along with the seed value from the user. The user was given an option to generate up to 5 random numbers and not more.
   Example: random number = (((time(0)*seed)*56)/9)%12

Results

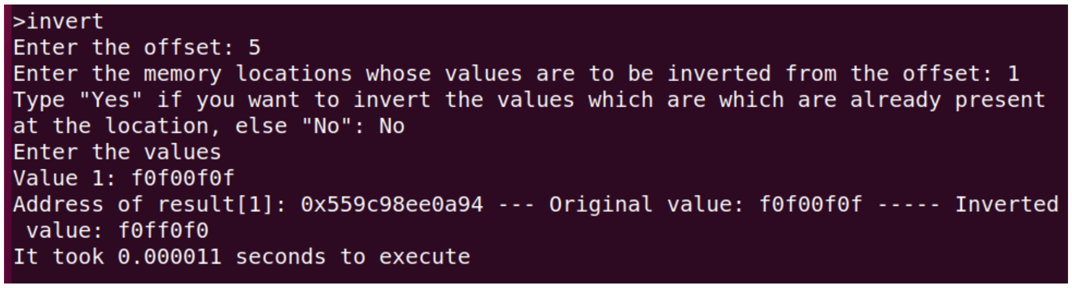
Following are the screenshots which show the sample result of various functions supported by the program:

- Allocating, writing and displaying functions



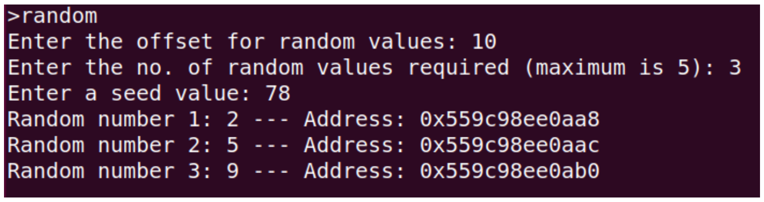
```
>allo
You are allocating memory
Enter number of elements: 100
100 memory locations allocated
Allocation address: 0x559c98ee0a80

>write
Specify the offset: 5
Specify the value you need to write in hexadecimal: 789f
Address: 0x559c98ee0a94 --- Value: 789f

>display
Specify the offset: 5
Specify the number of data locations: 2
Adress: 0x559c98ee0a94 --- Value: 789f
Adress: 0x559c98ee0a98 --- Value: 0
```

- Invert function



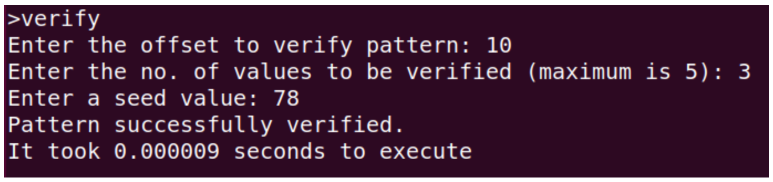
```
>invert
Enter the offset: 5
Enter the memory locations whose values are to be inverted from the offset: 1
Type "Yes" if you want to invert the values which are which are already present
at the location, else "No": No
Enter the values
Value 1: f0f00f0f
Address of result[1]: 0x559c98ee0a94 --- Original value: f0f00f0f ----- Inverted
 value: f0ff0f0
It took 0.000011 seconds to execute
```

- Random function



```
>random
Enter the offset for random values: 10
Enter the no. of random values required (maximum is 5): 3
Enter a seed value: 78
Random number 1: 2 --- Address: 0x559c98ee0aa8
Random number 2: 5 --- Address: 0x559c98ee0aac
Random number 3: 9 --- Address: 0x559c98ee0ab0
```

- Verify pattern



```
>verify
Enter the offset to verify pattern: 10
Enter the no. of values to be verified (maximum is 5): 3
Enter a seed value: 78
Pattern successfully verified.
It took 0.000009 seconds to execute
```

References

[1]. https://stackoverflow.com

[2]. https://www.programiz.com/c-programming/c-dynamic-memory-allocation

[3]. https://mathoverflow.net/questions/29494/pseudo-random-number-generation-algorithms

[4]. https://www.le.ac.uk/users/rjm1/cotter/page_59.htm

[5]. https://www.geeksforgeeks.org/clear-console-c-language/

[6]. https://study.com/academy/lesson/what-is-memory-access-violation.html