CS348-Assignment 1

Vatsal Gupta
200101105

README

20th January 2023

- **Name:** Vatsal Gupta
- **Roll No:** 200101105

Submission Contents

- README
- A1A_200101105.asm
- A1B_200101105.asm
- runner_gcc.sh ★(*provided extra i.e. not needed, provided for ease!*)
- Assignment 1_SET A.pdf
- Assignment 1_SET B.pdf
- assgn 1 alottment.pdf

runner_gcc.sh ★

```
foo=$1
fooplus=''
for (( i=0; i<${#foo}-4; i++ )); do
 fooplus+=${foo:$i:1}
done
# nasm -felf64 $fooplus.asm && ld $fooplus.o && ./a.out
nasm -felf64 $fooplus.asm && gcc -no-pie $fooplus.o && ./a.out
```

This is a shell script used to directly run the asm file in any question use it as:

1. `vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm`

2. `vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1B_200101105.asm`

Kind Note: make sure you give the shell script permissions by using chmod +x runner_gcc.sh beforehand.

Question 1

- **Files :** A1A_200101105.asm
- **System Version:** Linux (Ubuntu 20.04 LTS), NASM version 2.14.02

Key Points/Prerequisites

- All inputs are assumed to be integral in the 4 byte range. (and more specifically **n>=0,k>=0**)
- To run the code, simply use the runner_gcc.sh as above ***OR*** follow the below steps:

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ nasm -felf64 A1A_200101105.asm
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ gcc -no-pie A1A_200101105.o
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./a.out
```

- The code is well commented and involves the following basic steps:
    - TAKING INPUT N AND STORING IT IN “length” VARIABLE
    - TAKING INPUT K AND STORING IT IN k VARIABLE
    - COPYING K TO K_PRESERVE , to remember the value of k
    - SETTING FLAG AND K as needed in the question
    - TAKING ARRAY INPUT IN A LOOP
    - CALCULATING SUM UNTIL K
    - OUTPUTTING SUM AND FLAG
- The steps are shown in the comments of the code and the working and flow is explained as well
- The user is asked to enter the length of array
- Then the user is asked to enter k and k is copied to k_preserve
- If k>n flag is set to 0 and k is set to n, else flag is 1 and k is left as it is.
- User is asked to “enter array numbers” line after line
- The sum is calculated until k
- The flag and sum is outputted and a message is shown if flag is 0 saying that “k” (from k_preserve) numbers dont exist in the array.

Screenshot of output under different test cases:

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 5
Enter k: 3

Enter array numbers:
-10
42
-18
7
11
Sum=14
Flag=1
```

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 10
Enter k: 7

Enter array numbers:
1
3
5
10
20
79
8
3
11
6
Sum=126
Flag=1
```

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 6
Enter k: 6

Enter array numbers:
1
2
3
4
5
6
Sum=21
Flag=1
```

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 3
Enter k: 4

Enter array numbers:
1
90
32
Sum=123
Flag=0    ##4 numbers are not present in array
```

Corner Cases:

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 0
Enter k: 20

Enter array numbers:
Sum=0
Flag=0    ##20 numbers are not present in array
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1A_200101105.asm
Enter length of array(n): 0
Enter k: 0

Enter array numbers:
Sum=0
Flag=1
```

Question 2

- **Files :** A1B_200101105.asm
- **System Version:** Linux (Ubuntu 20.04 LTS), NASM version 2.14.02

Key Points/Prerequisites

- All inputs are assumed to be integral in the 4 byte range.(specifically, n>0,arr[i]>=0(preferably))
- To run the code, simply use the runner_gcc.sh as above ***OR*** follow the below steps:

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ nasm -felf64 A1B_200101105.asm
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ gcc -no-pie A1B_200101105.o
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./a.out
```

- The code is well commented and involves the following basic steps:
    - TAKING INPUT N AND STORING IT IN “length” VARIABLE
    - TAKING ARRAY INPUT IN A LOOP
    - CORNER CASE CHECKING
    - CHECKING FOR DUPLICACY WHILE TAKING INPUT
    - CHECKING FOR PRIMALITY WHILE TAKING INPUT
    - OUTPUTTING ARRAY
- The steps are shown in the comments of the code and the working and flow is explained as well
- The user is asked to enter the length of array
- User is asked to “enter array numbers” line after line
- Each time a number is entered, it is admitted into the array at i^{th} position (taking 8 bytes as a qword)
- The position at which the number is entered (i) is determined by the r12 register
- Once a number is read, we check whether the number is less than 2 (0,1 are the only possibilities) if so we discard it and re-read the new number at i^{th} position without changing the value of i.
- We then check by looping through all j<i whether the number has already been admitted into the array, if so we discard it and re-read the new number at i^{th} position without changing the value of i.
- We then check by looping through all j<arr[i] whether the number is perfectly divisible by any j, if so it is not prime and so we discard it and re-read the new number at i^{th} position without changing the value of i.
- Finally an array of “length” value is displayed containing prime, non duplicate elements.

Screenshot of output under different test cases:

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1B_200101105.asm
Enter length of array(n): 4

Enter array numbers:
4
5
9
10
13
829
947
Printing array of 4 elements
5 13 829 947
```

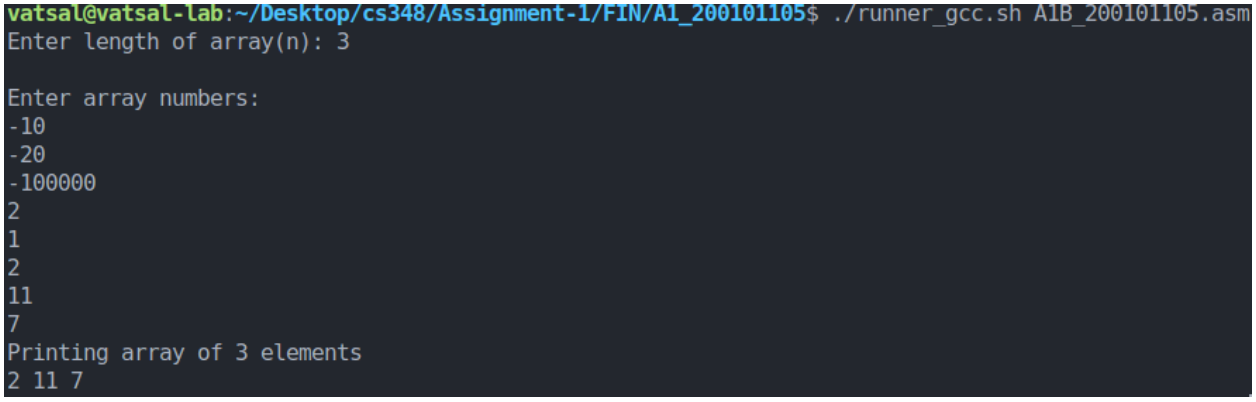
```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1B_200101105.asm
Enter length of array(n): 6

Enter array numbers:
1
2
2
2
3
5
5
5
331
337
337
331
2
5
719
Printing array of 6 elements
2 3 5 331 337 719
```

```
vatsal@vatsal-lab:~/Desktop/cs348/Assignment-1/FIN/A1_200101105$ ./runner_gcc.sh A1B_200101105.asm
Enter length of array(n): 5

Enter array numbers:
6037
2
80
10
10
10
10
80
2
6037
6037
2
2
6037
4409
11
4409
11
11
4409
7879
Printing array of 5 elements
6037 2 4409 11 7879
```

Corner Cases:



Constraints:

Q1:

Array size and type restrictions are as follows:

1. 500>n>=0,INT_MAX>k>=0 is necessary (INT_MAX as in C++14)
2. All arr[i] i.e array values must be of 4 byte range and of integer format and >=0(though some *negative* case handling as in corner cases is handled)
3. No more than 500 values are acceptable (though this can be easily modified by changing the code's *.data* section).

Q2:

Array size and type restrictions are as follows:

1. 1250>n>=0
2. All arr[i] i.e array values must be of 4 byte range and of integer format and >=0(though some *negative* case handling as in corner cases is handled)
3. No more than 1250 values are acceptable into the array (though this can be easily modified by changing the code's *.data* section).

Note:

1. User program flow is self explanatory during execution and program works perfectly under the constraints and system requirements as mentioned in the corresponding sections.
2. Corner Case handling is as shown.
3. Rbp register is handled properly
4. Used registers are pushed and popped before scanf and printf calls
5. Gcc is needed to use scanf and printf appropriately which are declared under the .text with keyword *extern*
6. The standard practice for printf and scanf is defined in the comment section of the first call of these functions there after, these are assumed to be understood and followed as per the convention defined the format is stored in a variable defined in the .data section which is moved to the rdi register the variable called and stored in the rsi register is defined in the *.data* section for all printf calls and the variable in which the input is stored is defined in the *.bss* section for all scanf calls
7. For using scanf we always pass the address of the variable where we want to store the user input to the rsi register and for all printf calls , if we want to print a string we pass the address of the string but if we want to print a number we pass the numeric value of it by dereferencing the variable itself this is as per C++17 convention.
8. To check loops, print statements are written which may be uncommented if needed. For loop convention is followed for loop construction with jump statements.

THANKS.