CS6570 Assignment 1a - Report

Rachit Tibrewal (CS16B022)
E Santhosh Kumar (CS16B107)

Q1

The program takes a string (*str*) as input and prints the string along with the n^{th} Fibonacci number for a given local variable n. The Fibonacci numbers are computed by the function $fib(n)$ that recursively calls the functions $fib(n-1)$ and $fib(n-2)$.

Q2

The string *str* is allocated 50 bytes on the heap.Giving a longer input to gets results in segmentation fault as we are accessing memory which has not been allocated to the program.

Q3

Refer figure 1.

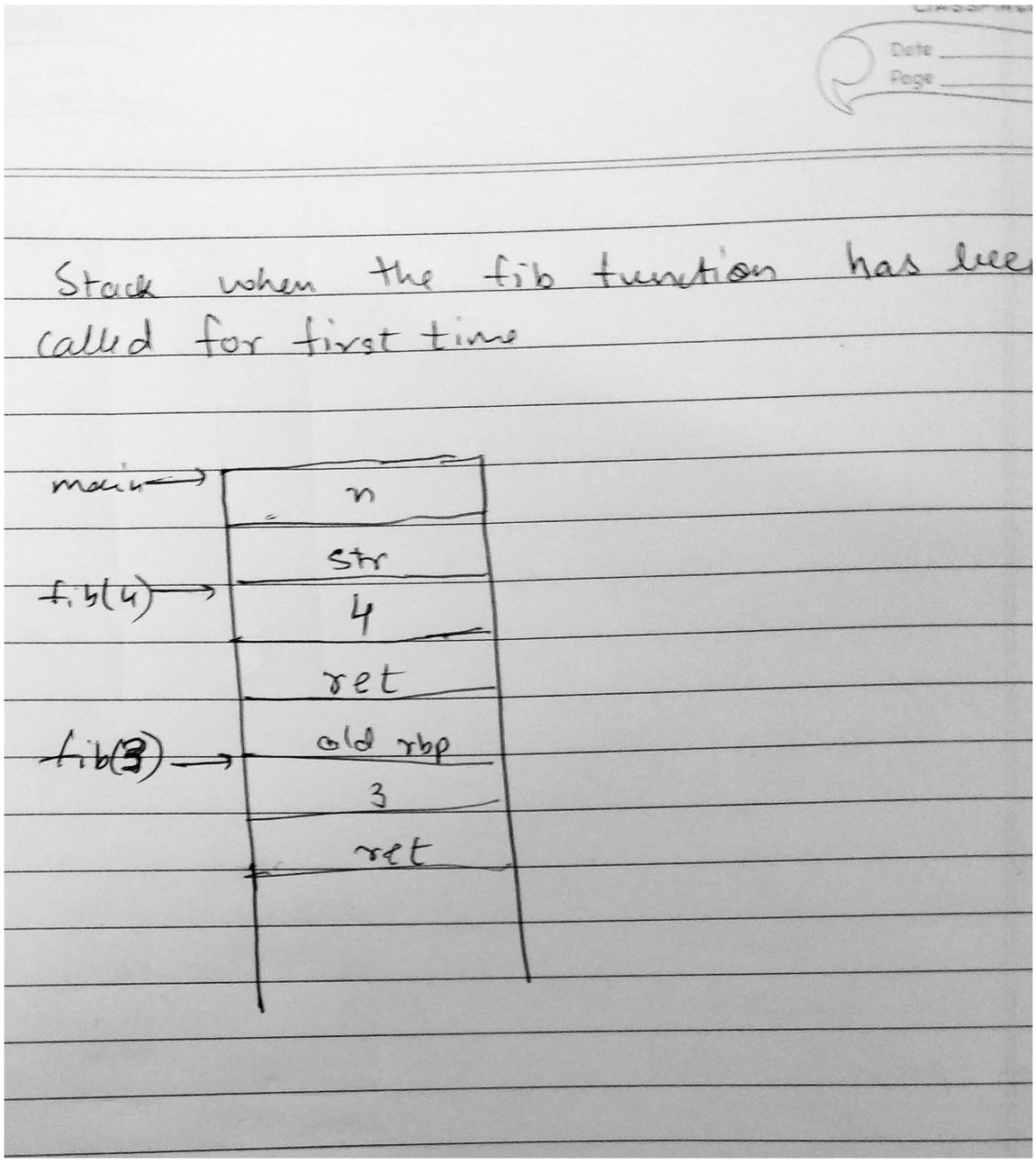

Figure 1: Q3

Q4

Refer figure 2.

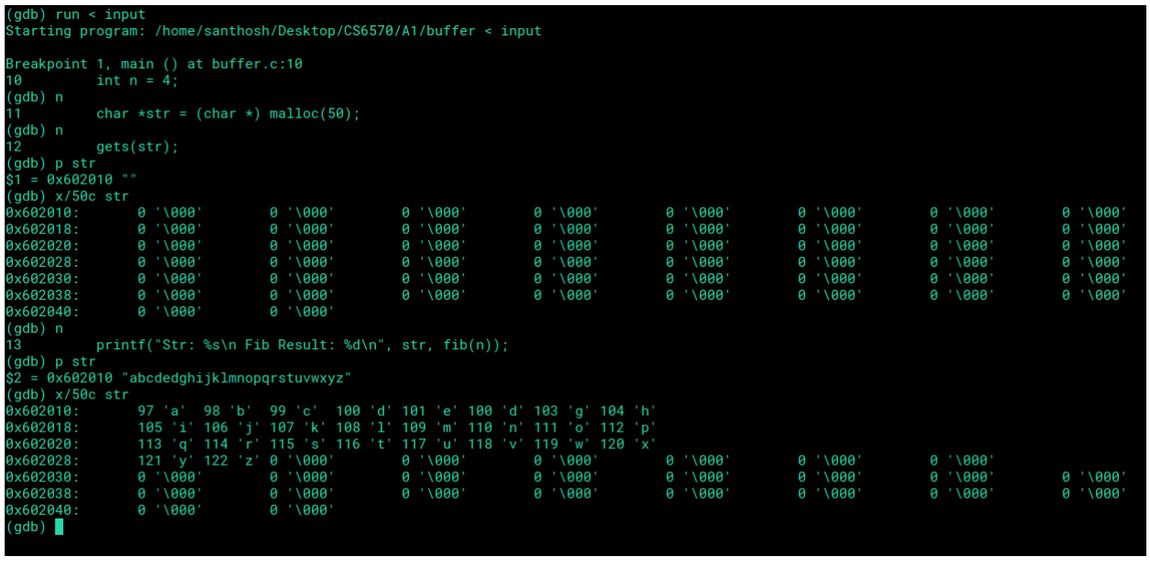

Figure 2: Q4

str points to address $0x602010$ on the heap. Before $gets(str)$ is executed, the 50 bytes on the heap starting from this address contain 0. Therefore string *str* is empty. Once the input string "abcdefghijklmnopqrstuvwxyz" is passed as input through *gets*, the first 26 bytes of *str* are written accordingly.

Q5

Refer figure 3.

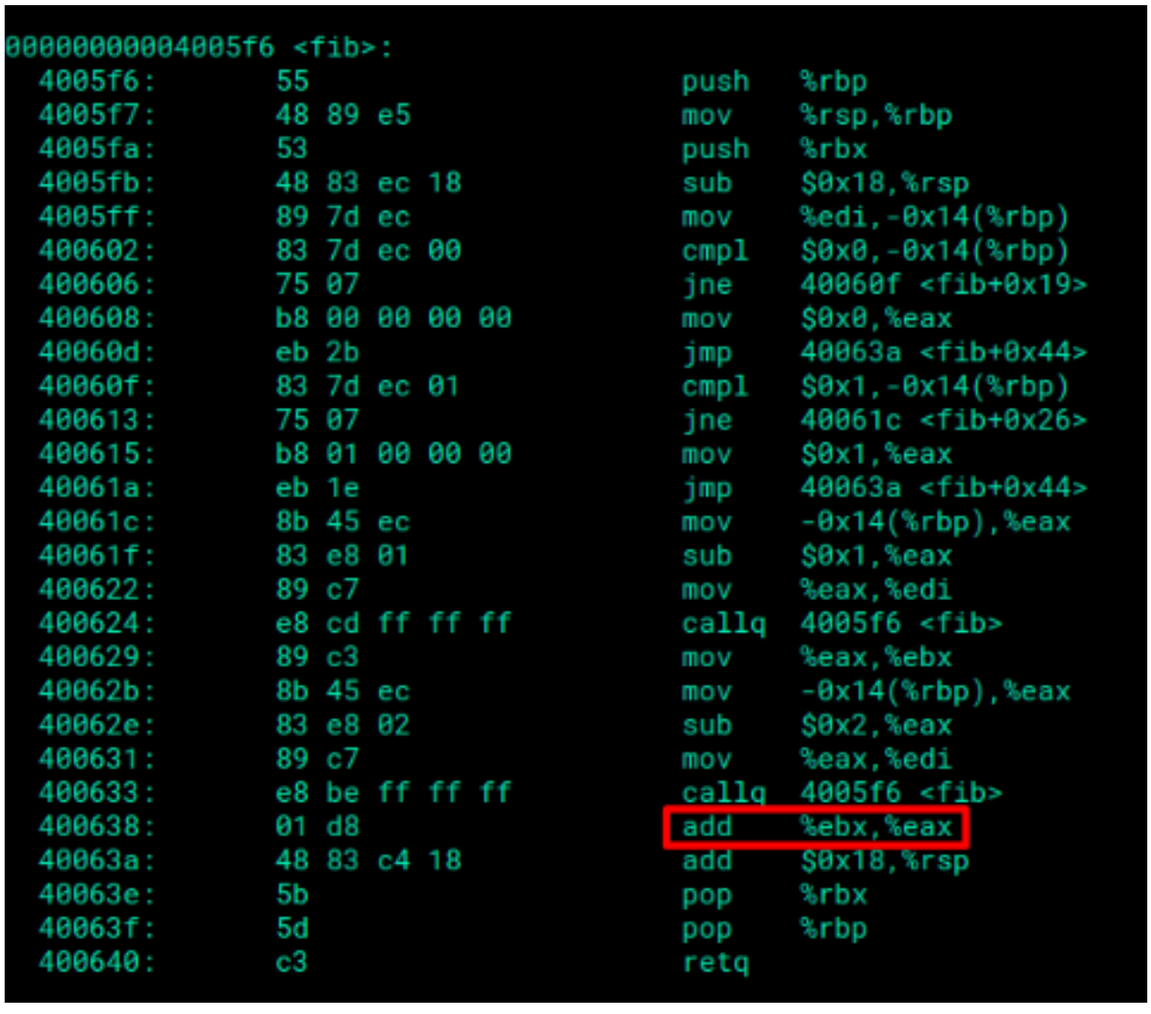

Figure 3: Q5

ebx and *eax* are the registers used to add $fib(n-1)$ and $fib(n-2)$. *ebx* stores the return value of $fib(n-1)$, *eax* stores the return value of $fib(n-2)$, ans the resulting sum is stored in *eax*.

Q6

We have modified the program to memoize previously computed fibonacci values. This doesn't affect the stack structure but reduces the runtime complexity significantly. We found that for $n = 174570$ stack overflow occurs for the first time. Investigating the assembly code, we get that each function frame is of 48 bytes and the main function's frame takes 24 bytes which is negligible compared to stack occupiedd due to recursive function calls. Maximum stack size occupied at anytime during the execution of the proram is thus 48 * n.

$$Stacksize = 48 * 174570 = 8379360 \approx 8MB$$

Listing 1: Modified C program

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int memo[1000000];
int fib(int n){
    if (memo[n]!= 1) return memo[n];
    if( n == 0 ) return 0;
    if( n == 1 ) return 1;
    return memo[n] = ( fib(n 1)  +  fib(n 2) );
}

int main( ){
    // n = 174570 : segmentation fault
    memset(memo, 1, sizeof(memo));
    int n = 174570;
    char *str = (char *) malloc(50);
    gets(str);
    printf("Str: %s\n Fib Result: %d\n", str, fib(n));
    free(str);
    return 0;
}
```

The frame size is 8 bytes for return address, 8 bytes due to pushq %rbp, 8 bytes due to pushq %rbx, 24 bytes due to subq $24, %rsp. No arguments are pushed, instead the argument is passed through register %edi. Thus, each function has frame size of 48 bytes.

Listing 2: Fib function assembly code

```
fib:
.LFB5:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, 16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        pushq   %rbx
        subq    $24, %rsp
        .cfi_offset 3, 24
        movl    %edi, 20(%rbp)
        movl    20(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        leaq    memo(%rip), %rax
        movl    (%rdx,%rax), %eax
        cmpl    $1, %eax
        je      .L2
        movl    20(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        leaq    memo(%rip), %rax
        movl    (%rdx,%rax), %eax
        jmp     .L3
```

The frame size is 8 bytes for pushq %rbp, 16 bytes for subq $16, %rsp. So in total the frame size for main function is 24 bytes.

Listing 3: Main function assembly code

```
main:
```

```
.LFB6:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, 16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $4000000, %edx
        movl    $1, %esi
        leaq    memo(%rip), %rdi
        call    memset@PLT
        movl    $174570, 12(%rbp)
        movl    $50, %edi
        call    malloc@PLT
        movq    %rax, 8(%rbp)
        movq    8(%rbp), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    gets@PLT
        movl    12(%rbp), %eax
        movl    %eax, %edi
        call    fib
        movl    %eax, %edx
        movq    8(%rbp), %rax
        movq    %rax, %rsi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movq    8(%rbp), %rax
        movq    %rax, %rdi
        call    free@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Q7

To make the stack size unlimited (limited by hardware memory), we shoudl use the command ulimit -s unlimited.