

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 get results in segmentation fault as we are accessing memory which has not been allocated to the program.

Q3

Refer figure 1.

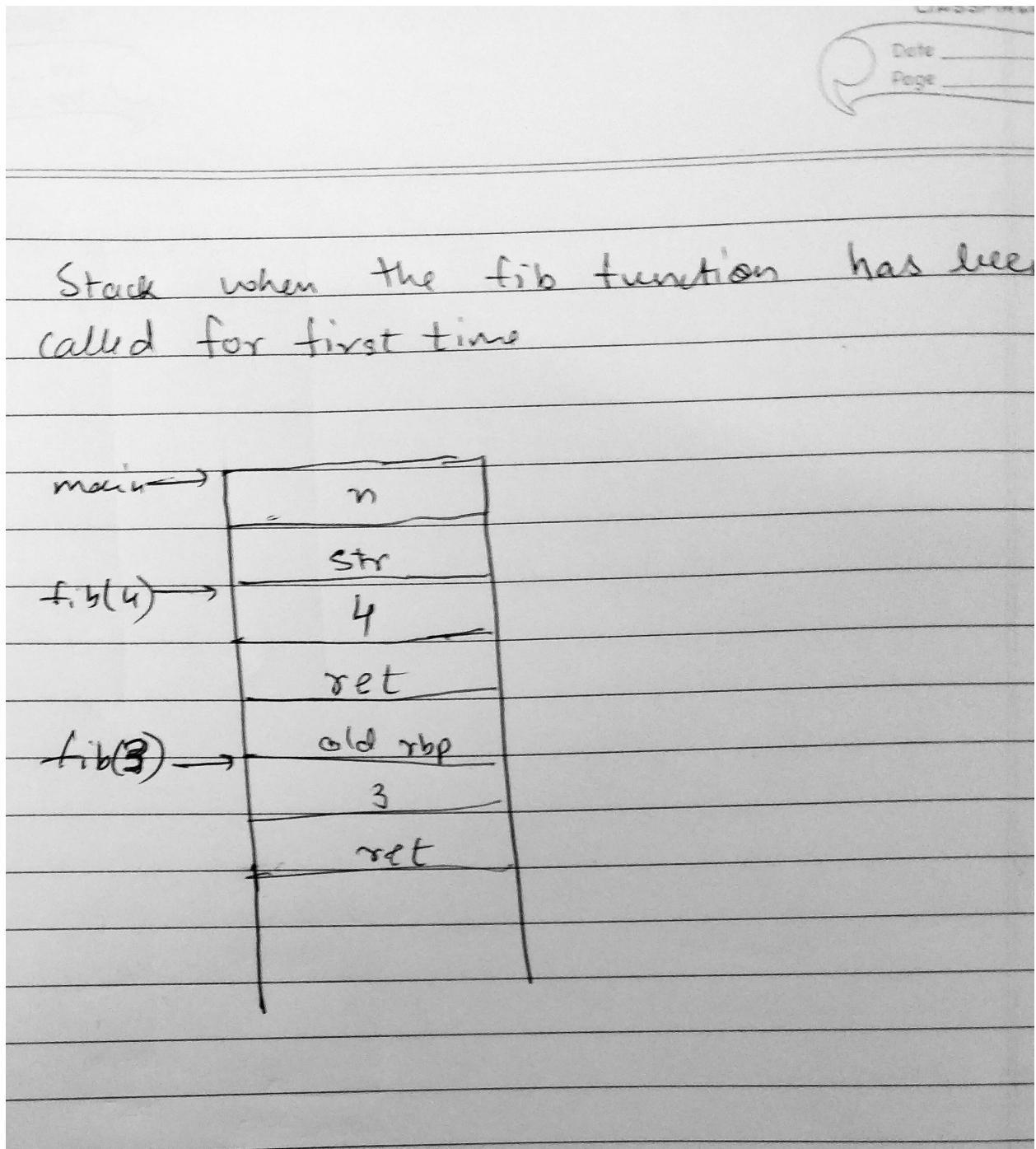


Figure 1: Q3

Q4

Refer figure 2.

```
(gdb) run < input
Starting program: /home/santhosh/Desktop/CS6570/A1/buffer < input

Breakpoint 1, main () at buffer.c:10
10      int n = 4;
(gdb) n
11      char *str = (char *) malloc(50);
(gdb) n
12      gets(str);
(gdb) p str
$1 = 0x602010 ""
(gdb) x/50c str
0x602010: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602018: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602020: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602028: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602030: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602038: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602040: 0 '\000' 0 '\000'
(gdb) n
13      printf("Str: %s\n Fib Result: %d\n", str, fib(n));
(gdb) p str
$2 = 0x602010 "abcdeghijklmnopqrstuvwxyz"
(gdb) x/50c str
0x602010: 97 'a' 98 'b' 99 'c' 100 'd' 101 'e' 102 'f' 103 'g' 104 'h'
0x602018: 105 'i' 106 'j' 107 'k' 108 'l' 109 'm' 110 'n' 111 'o' 112 'p'
0x602020: 113 'q' 114 'r' 115 's' 116 't' 117 'u' 118 'v' 119 'w' 120 'x'
0x602028: 121 'y' 122 'z' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602030: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602038: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x602040: 0 '\000' 0 '\000'
(gdb) █
```

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 "abcdeghijklmnopqrstuvwxyz" is passed as input through *gets*, the first 26 bytes of *str* are written accordingly.

Q5

Refer figure 3.

```
0000000004005f6 <fib>:
4005f6: 55          push    %rbp
4005f7: 48 89 e5    mov     %rsp,%rbp
4005fa: 53          push    %rbx
4005fb: 48 83 ec 18  sub    $0x18,%rsp
4005ff: 89 7d ec    mov     %edi,-0x14(%rbp)
400602: 83 7d ec 00  cmpl   $0x0,-0x14(%rbp)
400606: 75 07       jne     40060f <fib+0x19>
400608: b8 00 00 00 00 mov    $0x0,%eax
40060d: eb 2b       jmp     40063a <fib+0x44>
40060f: 83 7d ec 01  cmpl   $0x1,-0x14(%rbp)
400613: 75 07       jne     40061c <fib+0x26>
400615: b8 01 00 00 00 mov    $0x1,%eax
40061a: eb 1e       jmp     40063a <fib+0x44>
40061c: 8b 45 ec    mov     -0x14(%rbp),%eax
40061f: 83 e8 01    sub     $0x1,%eax
400622: 89 c7       mov     %eax,%edi
400624: e8 cd ff ff ff callq  4005f6 <fib>
400629: 89 c3       mov     %eax,%ebx
40062b: 8b 45 ec    mov     -0x14(%rbp),%eax
40062e: 83 e8 02    sub     $0x2,%eax
400631: 89 c7       mov     %eax,%edi
400633: e8 be ff ff ff callq  4005f6 <fib>
400638: 01 d8       add     %ebx,%eax
40063a: 48 83 c4 18  add     $0x18,%rsp
40063e: 5b         pop     %rbx
40063f: 5d         pop     %rbp
400640: c3         retq
```

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 $fib(n-2)$, and 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 occupied due to recursive function calls. Maximum stack size occupied at anytime during the execution of the program 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 should use the command `ulimit -s unlimited`.