CS310-Lab1

Members-

1. B20121 Pushkar Patel
2. B20228 Saksham Kumar
3. B20238 Vikas Dangi

Program-

```c
#include<stdio.h>
#include<stdlib.h>

double PI=3.14;
long MOD=1e9+7;
int evenPrime=2;
int UNLUCKY=7;

int state;
char *name;
float curr;

int recurse(int it)
{
    if(it==1) return 1;
    int a,b;
    a=it/2; b=3*it+1;
    printf("add a=%lx, add b=%lx\n", (unsigned long int)&a, (unsigned long
int)&b);
    if(it%2) return recurse(b);
    else return recurse(a);
}

int main(int argc, char const *argv[])
{
    char *str1=(char *)malloc(sizeof(char)*10);
    char *str2=(char *)malloc(sizeof(char)*10);
    printf("_main @ %lx\n",(long unsigned int)&main);
    printf("PI,MOD,evenPrime,UNLUCKy @ %lx %lx %lx %lx\n",(long unsigned
int)&PI,(long unsigned int)&MOD,(long unsigned int)&evenPrime,(long
unsigned int)&UNLUCKY);
    printf("state,name,curr @ %lx %lx %lx\n",(long unsigned
int)&state,(long unsigned int)&name,(long unsigned int)&curr);
```

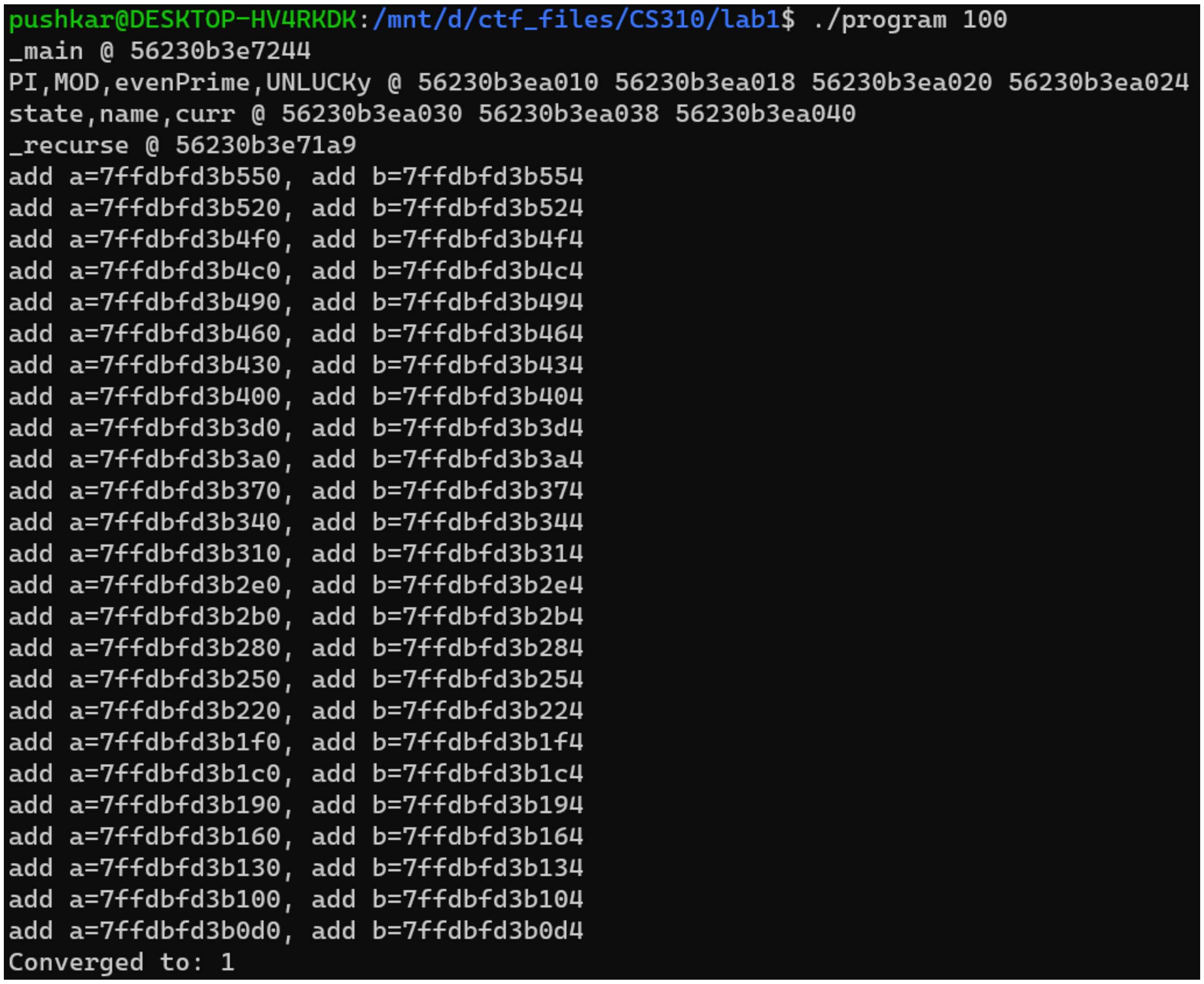
CS310-Lab1

```
    printf("_recurse @ %lx\n",(long unsigned int)&recurse);
    //printf("Argc= %d\n", argc);

    if(argc!=2) return 1;
    else
    {
        int a=atoi(argv[1]);

        printf("Converged to: %d\n",recurse(a));
    }
    return 0;
}
```

Q1. Which of the addresses printed by your program are defined in the output of objdump?
Ans - Addresses all global functions and variables are defined in output of objdump.

Q 2. Based on the output shown by the program, indicate what direction is the stack growing in [lower to higher addresses or vice-versa]?

CS310-Lab1



```
pushkar@DESKTOP-HV4RKDK:/mnt/d/ctf_files/CS310/lab1$ ./program 100
_main @ 56230b3e7244
PI,MOD,evenPrime,UNLUCKy @ 56230b3ea010 56230b3ea018 56230b3ea020 56230b3ea024
state,name,curr @ 56230b3ea030 56230b3ea038 56230b3ea040
_recurse @ 56230b3e71a9
add a=7ffdbfd3b550, add b=7ffdbfd3b554
add a=7ffdbfd3b520, add b=7ffdbfd3b524
add a=7ffdbfd3b4f0, add b=7ffdbfd3b4f4
add a=7ffdbfd3b4c0, add b=7ffdbfd3b4c4
add a=7ffdbfd3b490, add b=7ffdbfd3b494
add a=7ffdbfd3b460, add b=7ffdbfd3b464
add a=7ffdbfd3b430, add b=7ffdbfd3b434
add a=7ffdbfd3b400, add b=7ffdbfd3b404
add a=7ffdbfd3b3d0, add b=7ffdbfd3b3d4
add a=7ffdbfd3b3a0, add b=7ffdbfd3b3a4
add a=7ffdbfd3b370, add b=7ffdbfd3b374
add a=7ffdbfd3b340, add b=7ffdbfd3b344
add a=7ffdbfd3b310, add b=7ffdbfd3b314
add a=7ffdbfd3b2e0, add b=7ffdbfd3b2e4
add a=7ffdbfd3b2b0, add b=7ffdbfd3b2b4
add a=7ffdbfd3b280, add b=7ffdbfd3b284
add a=7ffdbfd3b250, add b=7ffdbfd3b254
add a=7ffdbfd3b220, add b=7ffdbfd3b224
add a=7ffdbfd3b1f0, add b=7ffdbfd3b1f4
add a=7ffdbfd3b1c0, add b=7ffdbfd3b1c4
add a=7ffdbfd3b190, add b=7ffdbfd3b194
add a=7ffdbfd3b160, add b=7ffdbfd3b164
add a=7ffdbfd3b130, add b=7ffdbfd3b134
add a=7ffdbfd3b100, add b=7ffdbfd3b104
add a=7ffdbfd3b0d0, add b=7ffdbfd3b0d4
Converged to: 1
```

The stack first grows from lower to higher addresses then while returning it returns from higher to lower addresses.

Q3. How large is the stack frame for each recursive call? (Google and find out what is a stackframe)
Depending upon the architecture of the machine, for the given address size of 64 bits, the address of the stack cannot exceed 64 bits in size.

Q4. Where is the heap? What direction is it growing in?
Whenever a call to malloc is made, memory is assigned from something called a heap. Heap is a large pool of memory that can be used dynamically – it is also known as the "free store". This is memory that is not automatically managed – you have to explicitly allocate (using functions such as malloc), and deallocate (e.g. free) the memory. .

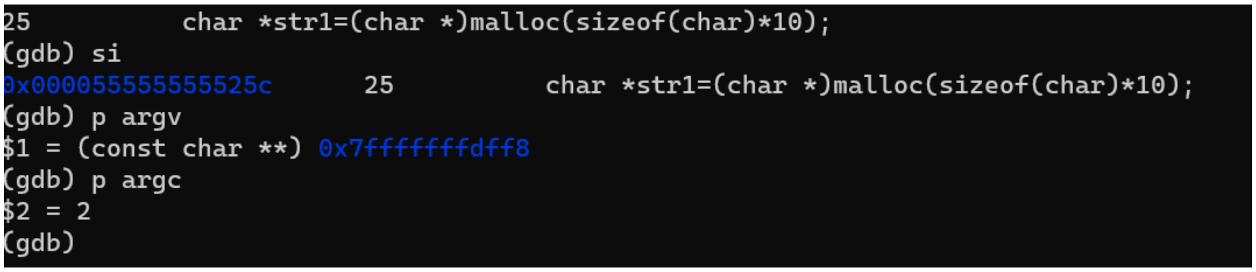
Q5. Are the two malloc()ed memory areas contiguous? (e.g. is there any extra space between their addresses?)

CS310-Lab1

No, the malloced memory areas are not contiguous, for contiguous memory allocation one might use calloc()ed memory.

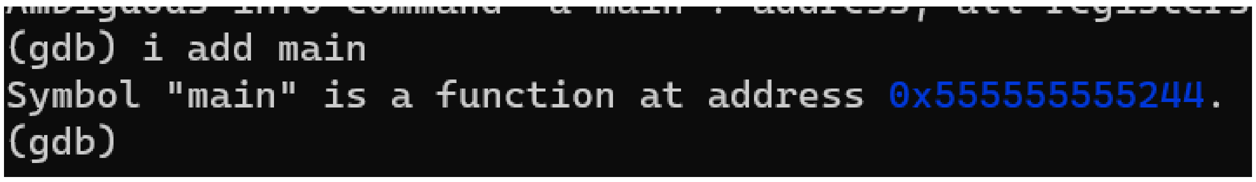
Q6. Load up your program executable in gdb, set a breakpoint at main, start your program, and continue one line at a time until you are in the middle of your program's execution (i.e. after a few recursive calls have been made). Take a look at the stack using backtrace (bt). While you are looking through gdb, answer the following questions: (hint: print var-name)
(a) Value of argv?

(b) Value pointed to by argv?



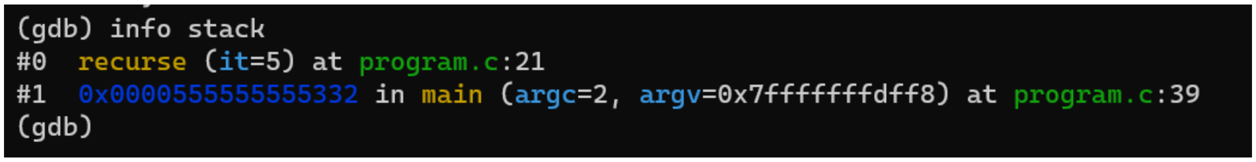
```
25          char *str1=(char *)malloc(sizeof(char)*10);
(gdb) si
0x000055555555525c      25          char *str1=(char *)malloc(sizeof(char)*10);
(gdb) p argv
$1 = (const char **) 0x7fffffffdff8
(gdb) p argc
$2 = 2
(gdb)
```

(c) Address of the function main? Type "info address main" .



```
Ambiguous info command "a main": address, all-registers
(gdb) i add main
Symbol "main" is a function at address 0x555555555244.
(gdb)
```

(d) Type "info stack" in gdb. Explain what you see.

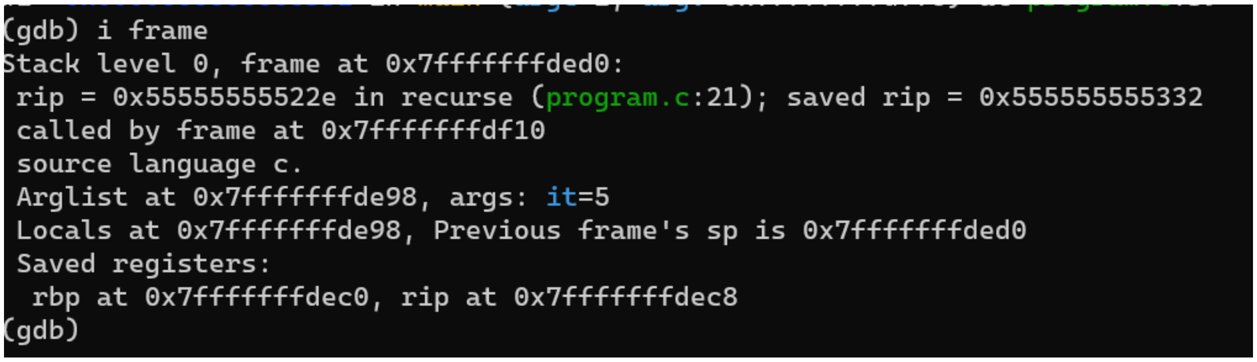


```
(gdb) info stack
#0  recurse (it=5) at program.c:21
#1  0x0000555555555332 in main (argc=2, argv=0x7fffffffdff8) at program.c:39
(gdb)
```

We can see the successive call backs in the stack frame. Callback of "recurse" function with parameter 5, from main at line 39.

(e) Type "info frame". Explain what you see.

CS310-Lab1



```
(gdb) i frame
Stack level 0, frame at 0x7fffffffded0:
 rip = 0x55555555522e in recurse (program.c:21); saved rip = 0x555555555332
 called by frame at 0x7fffffffdf10
 source language c.
 Arglist at 0x7fffffffde98, args: it=5
 Locals at 0x7fffffffde98, Previous frame's sp is 0x7fffffffded0
 Saved registers:
  rbp at 0x7fffffffdec0, rip at 0x7fffffffdec8
(gdb)
```

- frame num in backtrace, 0 is the current executing frame, which grows downwards, in consistency with the stack.
- The starting memory address of this stack frame is 0x7fffffffded0.
- Rip is the address of the next function to execute.
- Source language and starting address of the argument.

Q7. In the output of objdump, (a) What segment contains the main function and what is the address of main? (is it same as what you saw in gdb) (b) Do you see the stack segment anywhere? What about the heap? Explain.

a. .text function contains the main() function, address of main defined in objdump is 0x1244. It is not the same as gdb because when the program is being run it creates a virtual address space and adds an offset to start the program.
b. No, the stack and heap memory are initialized at the runtime of the program.