Name: Talise Baker-Matsuoka
ID: 7755662598
Email: bakermat@usc.edu

Fibonacci Parts 1 and 2

Program Desciption:

Fibonacci Part 1:
The purpose of this part of the lab is to compare the two methods we discussed in class for programs about Fibonacci
which are using recursion and using dynamic programming. As the numbers get larger, the recursive program takes increasingly longer
amounts of time to run when compared to the dynamic programming method.

Fibonacci Part 2:
In part 2, I implemented the exact same process as in lab 1 to run the executable file using python subprocesses instead. Then, I outputted
a statement about the results of running the c++ code.

How to run code:

Fibonacci Part 1:
For recursive:
$ g++ 7755662598fibonacci_recursive.cpp -o ./recursive.out
$ ./recursive.out 13

For dynamic programming:
$ g++ 7755662598fibonacci_dp.cpp -o ./dp.out
$ ./dp.out 13

Fibonacci Part 2:
$ python3 7755662598fibonacci.py

Notes:

For Fibonacci part 2, I used some background knowledge from my TAC class I took to help with the input stuff.

GDB and ASAN Parts 1-3

Program description:

GDB Part 1:

The purpose of this section is to learn how to use GDB to debug a broken program. The lab provides a broken program that must be debugged. The list of debugging commands and changes made are shown below.

Upon running the initial compilation using the command below, I get this warning. For now I am leaving it, but I will address it using GDB. I suspect this is (one of) the issue line(s).

```
7755662598Lab6_brokengdb1_correct.cpp:22:28: warning: unsequenced modification
and access to 'n' [-Wunsequenced]
   22 |         return  factorial(n--)*n;
      |                           ^   ~
1 warning generated.
```

After starting gdb using the command under the “how to run section” I began debugging.

Steps:

1. Add a break inside the factorial function

```
(gdb) break factorial
Breakpoint 1 at 0xc70: file 7755662598Lab6_brokengdb1_correct.cpp, line 21.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000000c70 in factorial(int)
                                                   at 7755662598Lab6_brokengdb1_correct.cpp:21
(gdb)
```

2. Run the program with a test number

```
(gdb) run
Starting program: /home/bakermat/ee355_lab6/brokengdb.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".
Please enter n:
5

Breakpoint 1, factorial (n=5) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb)
```

3. Now, step through the function to get to the suspected problem line

```
(gdb) s
22              return  factorial(n--)*n;
(gdb) l
17
18      long factorial(int n)
19      {
20
21          if(n)
22              return  factorial(n--)*n;
23          else return 1;
24
25
26      }
(gdb)
```

4. Print out the variable n before the recursive call of factorial

```
(gdb) print n
$1 = 5
(gdb)
```

Here, it can be seen that n = 5 which makes sense

5. Step into factorial (the recursive call of it) and again check what n is equal to when factorial is called again

```
(gdb) s

Breakpoint 1, factorial (n=5) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) print n
$2 = 5
(gdb)
```

Here, again, n = 5 which is incorrect since this means that n is not decrementing and there is an infinite loop here which is bad.

6. Check one last time to confirm the issue

```
(gdb) s
22              return  factorial(n--)*n;
(gdb) s

Breakpoint 1, factorial (n=5) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) print n
$3 = 5
(gdb)
```

This confirms that there is an infinite loop and that n is not changing. From lecture, it is evident that the issue lies in the n-- command. This command decrements n after the function is called so the line “return factorial(n--)*n” is actually giving return factorial(5) * 4 with the input of 5. This line must be changed.

7. Confirm infinite recursion with backtrace command

```
(gdb) bt
#0  factorial (n=5) at 7755662598Lab6_brokengdb1_correct.cpp:21
#1  0x0000aaaaaaaa0c8c in factorial (n=4)
    at 7755662598Lab6_brokengdb1_correct.cpp:22
#2  0x0000aaaaaaaa0c8c in factorial (n=4)
    at 7755662598Lab6_brokengdb1_correct.cpp:22
#3  0x0000aaaaaaaa0bf8 in main () at 7755662598Lab6_brokengdb1_correct.cpp:12
(gdb)
```

8. Fix the line of code

```
18 long factorial(int n)
19 {
20
21     if(n)
22         return  n * factorial(n - 1);
23     else return 1;
24
25
26 }
```

9. Quit gdb and recompile

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4036] will be killed.

Quit anyway? (y or n) y
```

The code is recompiled with the commands from the how to run section

10. Rerun gdb and go through the same steps to make sure that n is decrementing properly

```
(gdb) b factorial
Breakpoint 1 at 0xc74: file 7755662598Lab6_brokengdb1_correct.cpp, line 21.
(gdb) run
Starting program: /home/bakermat/ee355_lab6/brokengdb.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".
Please enter n:
5

Breakpoint 1, factorial (n=5) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) s
22              return  n * factorial(n - 1);
(gdb) print n
$1 = 5
(gdb) s

Breakpoint 1, factorial (n=4) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) print n
$2 = 4
(gdb) s
22              return  n * factorial(n - 1);
(gdb) s

Breakpoint 1, factorial (n=3) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) print n
$3 = 3
(gdb) s
22              return  n * factorial(n - 1);
(gdb) s

Breakpoint 1, factorial (n=2) at 7755662598Lab6_brokengdb1_correct.cpp:21
21          if(n)
(gdb) print n
$4 = 2
(gdb)
```

11. Run the code and see that it is now working right!

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./brokengdb.out
Please enter n:
5
n factorial is: 120
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$
```

ASAN Part 2:

The purpose of this section of the lab is to learn to use ASan to help debug a code that has memory related errors. A demonstration of how this tool was used to debug the provided file is below.

1. Compile and run code based on the how to run section and look at the error

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./brokenASan
=================================================================
==30146==ERROR: AddressSanitizer: stack-buffer-overflow on address 0xfffffca4fe68 at pc 0xbd5fae250d48 bp 0xfffffca4fdb0 sp 0xfffffca4fda0
READ of size 4 at 0xfffffca4fe68 thread T0
    #0 0xbd5fae250d44 in partition(int*, int, int) /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:5
    #1 0xbd5fae25117c in quickSort(int*, int, int) /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:30
    #2 0xbd5fae2513ec in main /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:50
    #3 0xfb5d01f073fc in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #4 0xfb5d01f074d4 in __libc_start_main_impl ../csu/libc-start.c:392
    #5 0xbd5fae250bec in _start (/home/bakermat/ee355_lab6/brokenASan+0xbec)

Address 0xfffffca4fe68 is located in stack of thread T0 at offset 56 in frame
    #0 0xbd5fae251288 in main /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:47

  This frame has 1 object(s):
    [32, 56) 'arr' (line 48) <== Memory access at offset 56 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:5 in partition(int*, int, int)
Shadow bytes around the buggy address:
  0x200fff949f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949f90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949fa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x200fff949fc0: 00 00 00 00 00 00 f1 f1 f1 f1 00 00 00[f3]f3 f3
  0x200fff949fd0: f3 f3 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff949ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff94a000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff94a010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==30146==ABORTING
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$
```

2. Looking at this large amount of text, go step by step down the list

The type of error says “Memory access at offset 56 overflows this variable”, when referencing the arr variable so the goal is to identify where this occurs.

The error occurs in partition(int*, int, int) in line 5 and we can see that is an overflow error. This error can be traced back to quicksort in line 30 and then main in line 50. Looking in main, at like 50, we see the following call to quicksort:

```
46 int main()
47 {
48     int arr[] = {10, 7, 8, 9, 1, 5};
49     int n = sizeof(arr)/sizeof(arr[0]);
50     quickSort(arr, 0, n);
```

Then, looking at line 30, we see that quickSort calls partition as shown here:

```
26 void quickSort(int arr[], int low, int high)
27 {
28     if (low < high)
29     {
30         int pi = partition(arr, low, high);
```

Finally, in line 5, the error starts in the partition function:

```
3 int partition (int arr[], int low, int high)
4 {
5     int pivot = arr[high];    // pivot
6     int i = (low - 1);  // Index of smaller element
7     int temp = 0;
```

It appears therefore that the issue is with the int high value not being in the array based on the overflow error at line 5. Looking at the call in line 30, we see that the input to partition is the same as the input to quickSort. Finally, looking at line 50, we see the value that is the issue is the int n value. Based on the calculation of n, n = 6 here. However, the 6th element of arr[] does not exist since the index starts at 0. This must be corrected by subtracting 1 from n when calling quickSort to avoid this issue. The fixed code is here:

```
46 int main()
47 {
48     int arr[] = {10, 7, 8, 9, 1, 5};
49     int n = sizeof(arr)/sizeof(arr[0]);
50     quickSort(arr, 0, n - 1);
```

3. Recompile and rerun ASan using the commands in the how to run section. The following output appeared.

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ g++ -fsanitize=address -ggdb -o brokenASan 7755662598Lab6_brokenASan_correct.cpp
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./brokenASan
=================================================================
==3435==ERROR: AddressSanitizer: stack-buffer-overflow on address 0xfffffc5dcf38 at pc 0xb5123ec81238 bp 0xfffffc5dceb0 sp 0xfffffc5dcea0
READ of size 4 at 0xfffffc5dcf38 thread T0
    #0 0xb5123ec81234 in printArray(int*, int) /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:41
    #1 0xb5123ec8140c in main /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:52
    #2 0xfe7a252b73fc in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0xfe7a252b74d4 in __libc_start_main_impl ../csu/libc-start.c:392
    #4 0xb5123ec80bec in _start (/home/bakermat/ee355_lab6/brokenASan+0xbec)

Address 0xfffffc5dcf38 is located in stack of thread T0 at offset 56 in frame
    #0 0xb5123ec81288 in main /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:47

  This frame has 1 object(s):
    [32, 56) 'arr' (line 48) <== Memory access at offset 56 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/bakermat/ee355_lab6/7755662598Lab6_brokenASan_correct.cpp:41 in printArray(int*, int)
Shadow bytes around the buggy address:
  0x200fff8bb990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bb9a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bb9b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bb9c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bb9d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x200fff8bb9e0: f1 f1 f1 f1 00 00 00[f3]f3 f3 f3 f3 00 00 00 00
  0x200fff8bb9f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bba00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bba10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bba20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x200fff8bba30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==3435==ABORTING
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$
```

4. Look at this new error piece by piece

This error is the same type as the previous error but occurs in a new location which must be traced back to the source of the problem.

The error starts in printArray at line 41. Then, the error can be traced back to main in line 52. Looking at this line shows the following code:

```
45 // Driver program to test above functions
46 int main()
47 {
48     int arr[] = {10, 7, 8, 9, 1, 5};
49     int n = sizeof(arr)/sizeof(arr[0]);
50     quickSort(arr, 0, n - 1);
51     printf("Sorted array: n");
52     printArray(arr, n);
53     return 0;
54 }
```

My initial guess is that there is the same issue with the integer n, but to confirm I will look at line 41 in printArray:

```
36 /* Function to print an array */
37 void printArray(int arr[], int size)
38 {
39     int i;
40     for (i=0; i <= size; i++)
41         printf("%d ", arr[i]);
42     printf("n");
43 }
```

Based on line 41, the issue is that the int i variable is outside the size of the array. That is because the variable size in this function has the value of n = 6 from main as seen before. This can be corrected by applying the same n - 1 or using a < rather than a <= symbol. I am fixing the code as shown below:

```
36 /* Function to print an array */
37 void printArray(int arr[], int size)
38 {
39     int i;
40     for (i=0; i < size; i++)
41         printf("%d ", arr[i]);
42     printf("n");
43 }
```

5. Compile and run again as shown in the how to run section to make sure there are no more errors.

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ g++ -fsanitize=address -ggd
b -o brokenASan 7755662598Lab6_brokenASan_correct.cpp
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./brokenASan
Sorted array: n1 5 7 8 9 10 nbakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6
```

How to run:

GDB Part 1:
Compile:
$ g++ -g 7755662598Lab6_brokengdb1_correct.cpp -o ./brokengdb.out

Start gdb:
$ gdb ./brokengdb.out

Run code:
$ ./brokengdb.out

ASAN Part 2:
Compile with ASan:
$ g++ -fsanitize=address -ggdb -o brokenASan 7755662598Lab6_brokenASan_correct.cpp

Run code with ASan:
$ ./brokenASan