Karthikeyan G
CB.SC.P2CYS24008

Secure Coding Lab Experiment - 7
Memory Layout

Memory layout

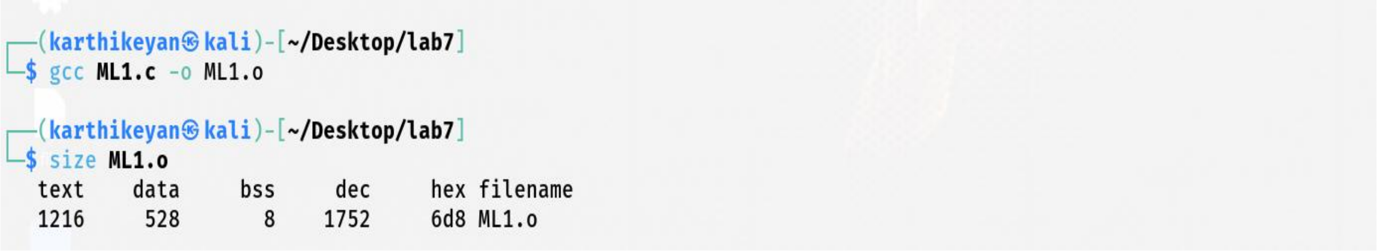
- **Stack:** Manages function calls and local variables, grows downwards in memory.
- **Heap**: Used for dynamic memory allocation, growing and shrinking as needed.
- **BSS Segment**: Holds uninitialized global and static variables, automatically set to zero.
- **Data Segment**: Stores initialized global and static variables.
- **Text Segment**: Contains the executable code of the program, usually read-only.

Size

The size command displays the sizes of the text, data, and BSS segments of a binary file or executable, helping to analyze its memory layout.

```
#include<stdio.h>
int main()
{
return 0;
}
```

Output:

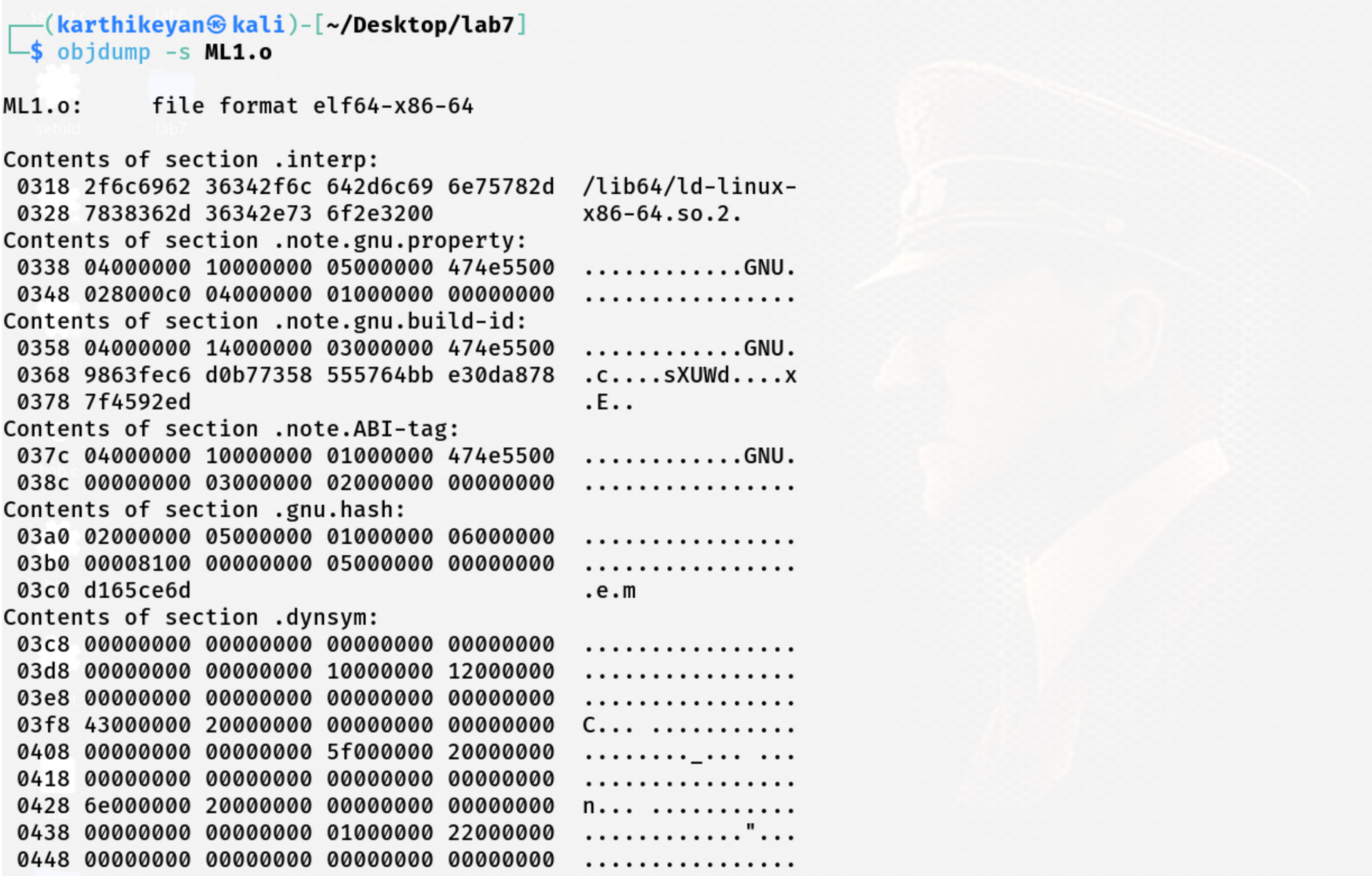


```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc ML1.c -o ML1.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML1.o
   text    data     bss     dec     hex filename
   1216     528       8    1752     6d8 ML1.o
```

Text/ code segment

Objdump

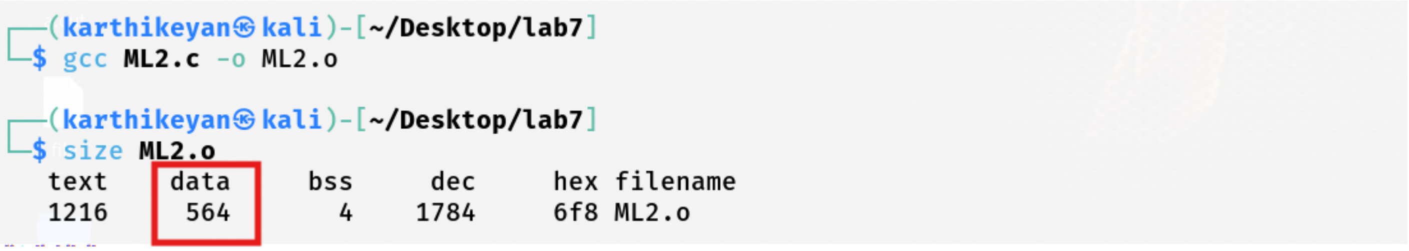
objdump is a command-line tool that displays information about object files, including assembly code, symbol tables, and section headers. It helps in analyzing and debugging compiled binaries by providing insights into their structure and contents.



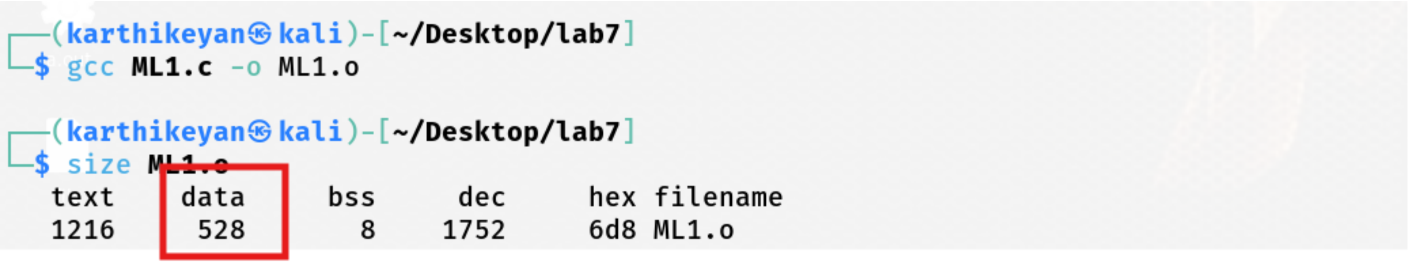
```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ objdump -s ML1.o

ML1.o:     file format elf64-x86-64

Contents of section .interp:
 0318 2f6c6962 36342f6c 642d6c69 6e75782d  /lib64/ld-linux-
 0328 7838362d 36342e73 6f2e3200           x86-64.so.2.
Contents of section .note.gnu.property:
 0338 04000000 10000000 05000000 474e5500  ............GNU.
 0348 028000c0 04000000 01000000 00000000  ................
Contents of section .note.gnu.build-id:
 0358 04000000 14000000 03000000 474e5500  ............GNU.
 0368 9863fec6 d0b77358 555764bb e30da878  .c....sXUWd....x
 0378 7f4592ed                             .E..
Contents of section .note.ABI-tag:
 037c 04000000 10000000 01000000 474e5500  ............GNU.
 038c 00000000 03000000 02000000 00000000  ................
Contents of section .gnu.hash:
 03a0 02000000 05000000 01000000 06000000  ................
 03b0 00008100 00000000 05000000 00000000  ................
 03c0 d165ce6d                             .e.m
Contents of section .dynsym:
 03c8 00000000 00000000 00000000 00000000  ................
 03d8 00000000 00000000 10000000 12000000  ................
 03e8 00000000 00000000 00000000 00000000  ................
 03f8 43000000 20000000 00000000 00000000  C... ...........
 0408 00000000 00000000 5f000000 20000000  ........_... ...
 0418 00000000 00000000 00000000 00000000  ................
 0428 6e000000 20000000 00000000 00000000  n... ...........
 0438 00000000 00000000 01000000 22000000  ............"...
 0448 00000000 00000000 00000000 00000000  ................
```

Initialized data segmentation

```
//pgm2.c

#include<stdio.h>
int a=10;
char ch='A';
int arr[5] = {1,2,3,4,5};
int main()
{
	return 0;
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc ML2.c -o ML2.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML2.o
   text    data     bss     dec     hex filename
   1216     564       4    1784     6f8 ML2.o
```

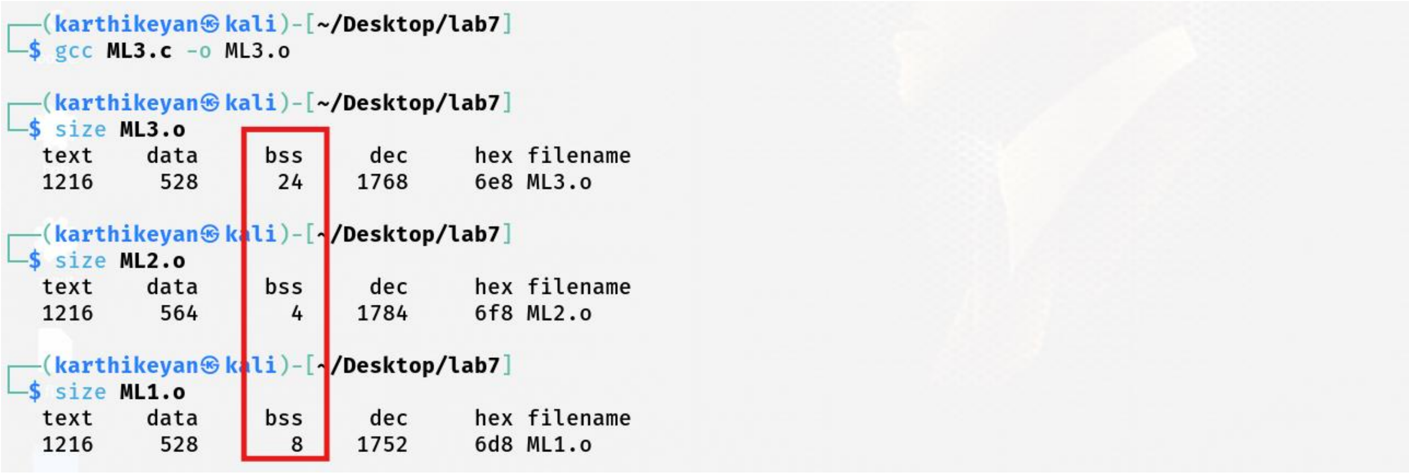


```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc ML1.c -o ML1.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML1.o
   text    data     bss     dec     hex filename
   1216     528       8    1752     6d8 ML1.o
```

The initialized data segment size increases due to the initialization of the variables (a, ch, and arr), the array's size, and potential padding for alignment.

Uninitialized data segmentation

```
//pgm3.c

#include<stdio.h>
int a,b,c;
char ch;
int main()
{
return 0;
}
```

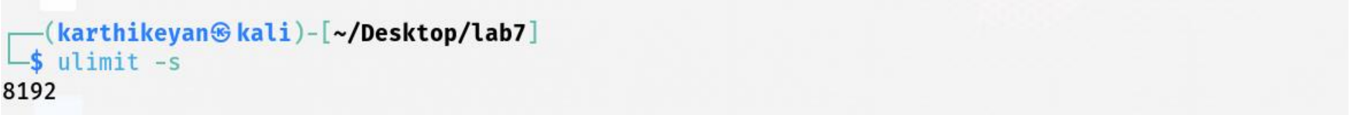


```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc ML3.c -o ML3.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML3.o
   text    data     bss     dec     hex filename
   1216     528      24    1768     6e8 ML3.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML2.o
   text    data     bss     dec     hex filename
   1216     564       4    1784     6f8 ML2.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size ML1.o
   text    data     bss     dec     hex filename
   1216     528       8    1752     6d8 ML1.o
```

The BSS segment in above case holds 24 bytes for uninitialized global variables, which are set to zero at program startup

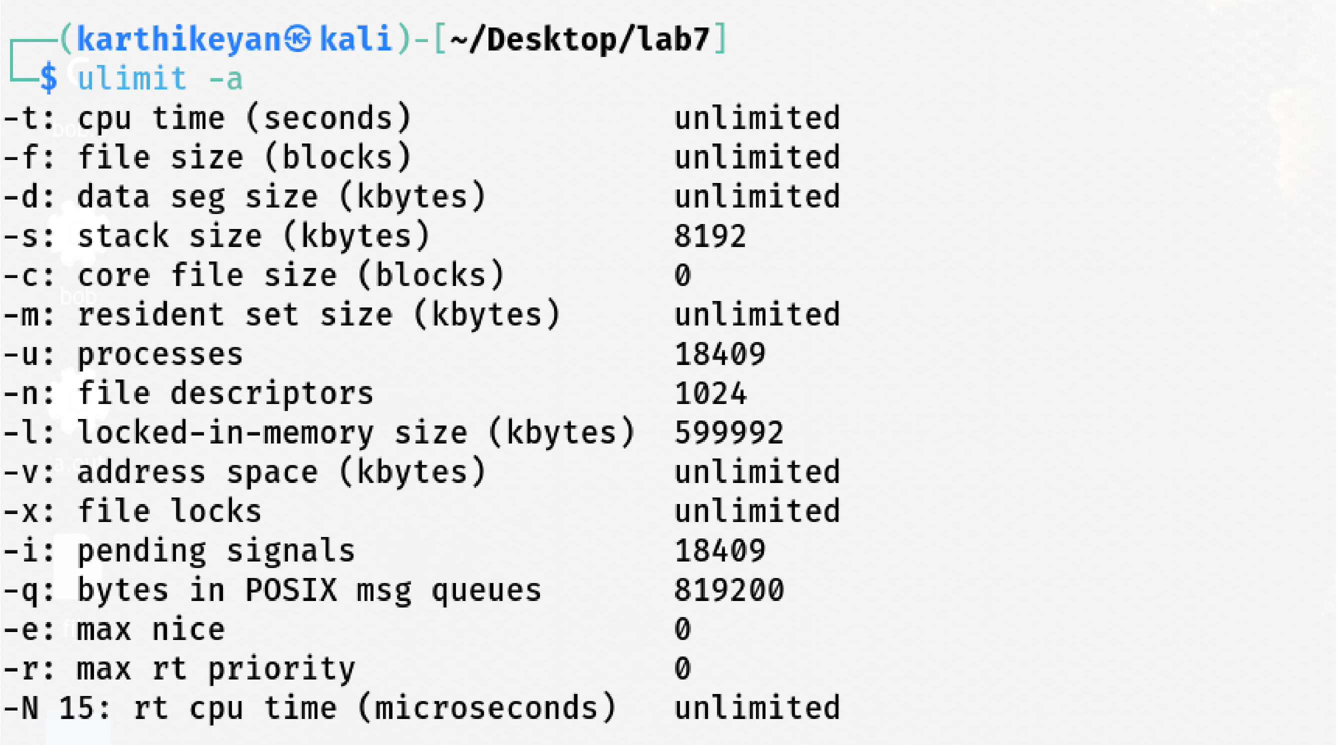
Stack

ulimit -s displays the stack size limit for the current user session.



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ulimit -s
8192
```

ulimit -a shows all resource limits for the current user session.

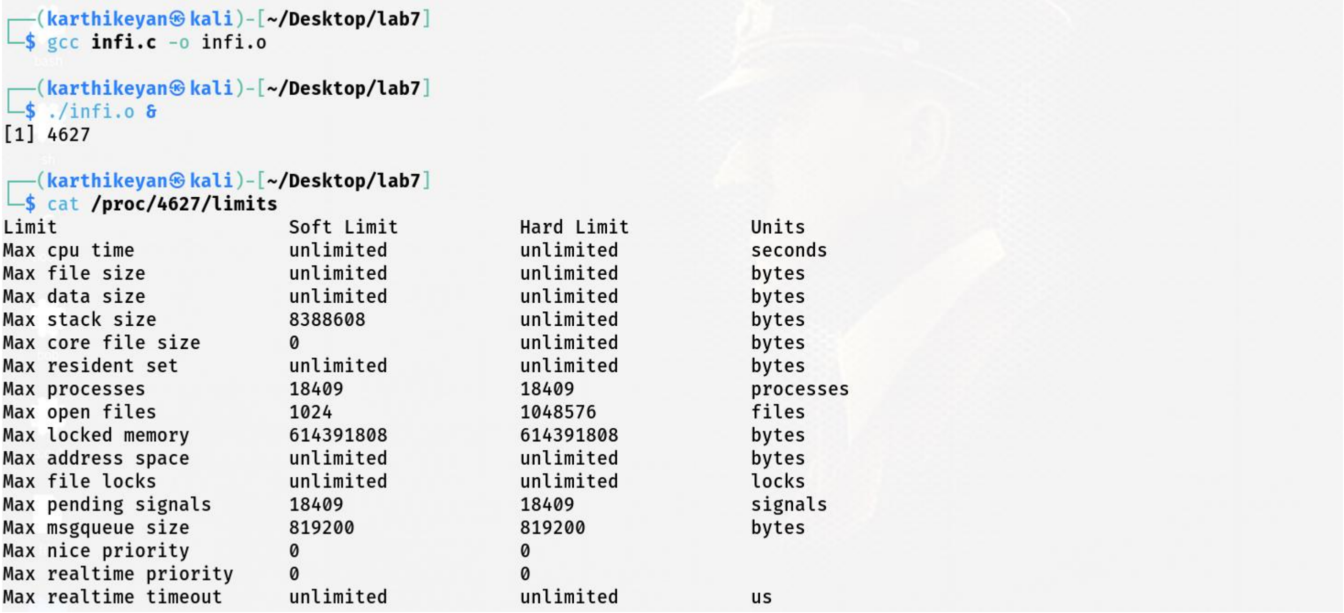


```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ulimit -a
-t: cpu time (seconds)              unlimited
-f: file size (blocks)              unlimited
-d: data seg size (kbytes)          unlimited
-s: stack size (kbytes)             8192
-c: core file size (blocks)         0
-m: resident set size (kbytes)      unlimited
-u: processes                       18409
-n: file descriptors                1024
-l: locked-in-memory size (kbytes)  599992
-v: address space (kbytes)          unlimited
-x: file locks                      unlimited
-i: pending signals                 18409
-q: bytes in POSIX msg queues       819200
-e: max nice                        0
-r: max rt priority                 0
-N 15: rt cpu time (microseconds)   unlimited
```

limits of a running process

cat /proc//limits command can be used to get the limits of a running process

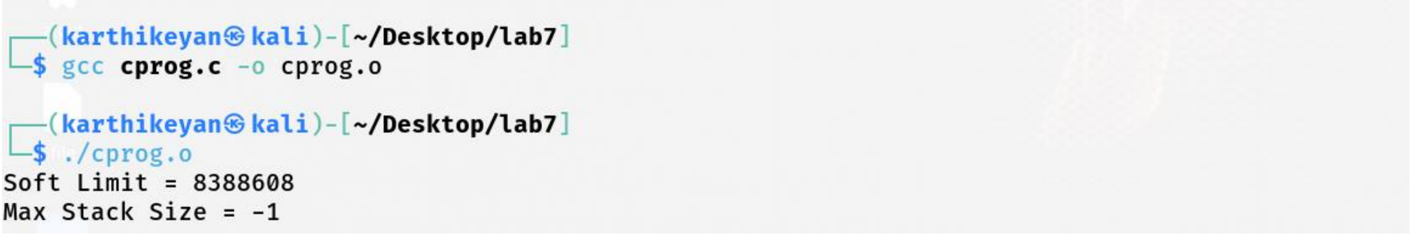
```
//infi.c

int main()
{
while(1){}
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc infi.c -o infi.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ./infi.o &
[1] 4627

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ cat /proc/4627/limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             18409                18409                processes
Max open files            1024                 1048576              files
Max locked memory         614391808            614391808            bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       18409                18409                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
```

Stack size using C program

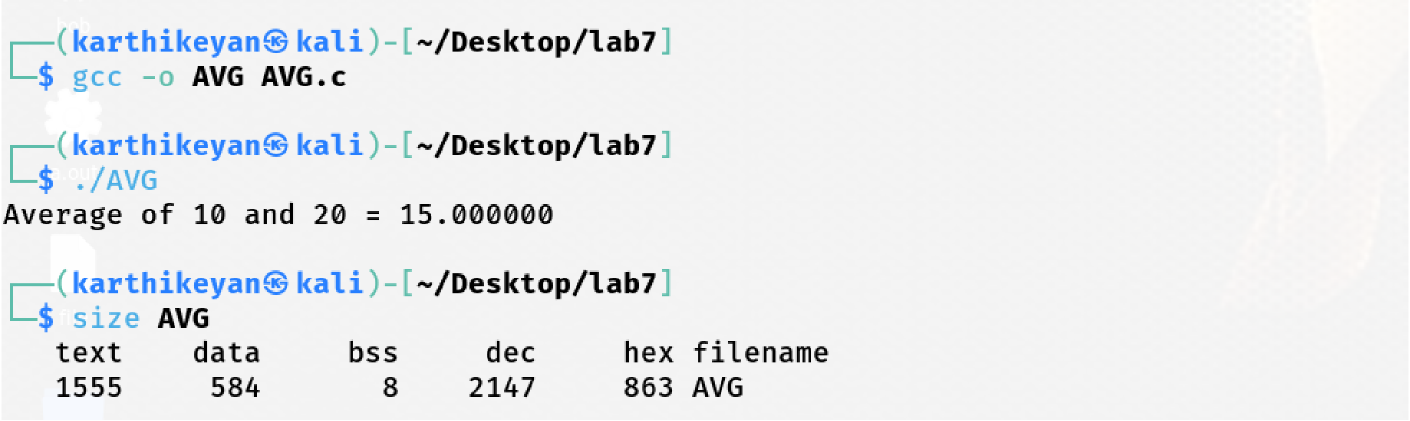
```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/resource.h>
int main()
{
struct rlimit lim;
if(getrlimit(RLIMIT_STACK,&lim)==0)
{
printf("Soft Limit = %ldn",lim.rlim_cur);
printf("Max Stack Size = %ldn",lim.rlim_max);
}
else
printf("%sn", strerror(errno));
return 0;
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc cprog.c -o cprog.o

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ./cprog.o
Soft Limit = 8388608
Max Stack Size = -1
```

Function

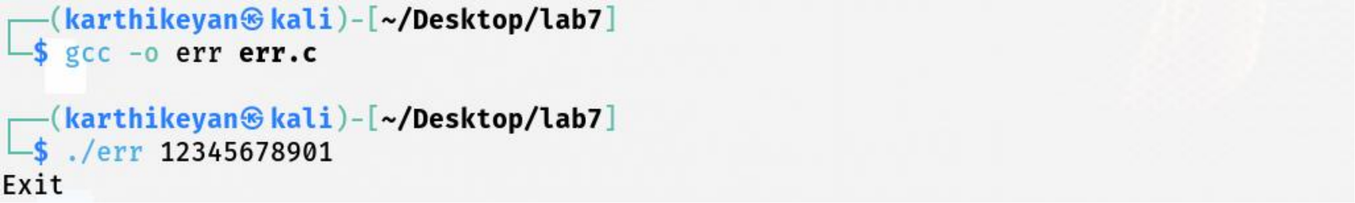
```
#include<stdio.h>

int sum(int a, int b)
{
        return a + b;
}

float avg(int a, int b)
{
        int s = sum(a, b);
        return (float)s / 2;
}

int main()
{
        int a = 10;
        int b = 20;
        printf("Average of %d, %d = %fn", a, b, avg(a, b));
        return 0;
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc -o AVG AVG.c

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ./AVG
Average of 10 and 20 = 15.000000

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ size AVG
   text    data     bss     dec     hex filename
   1555     584       8    2147     863 AVG
```

Stack error conditions (Unbounded string copy)

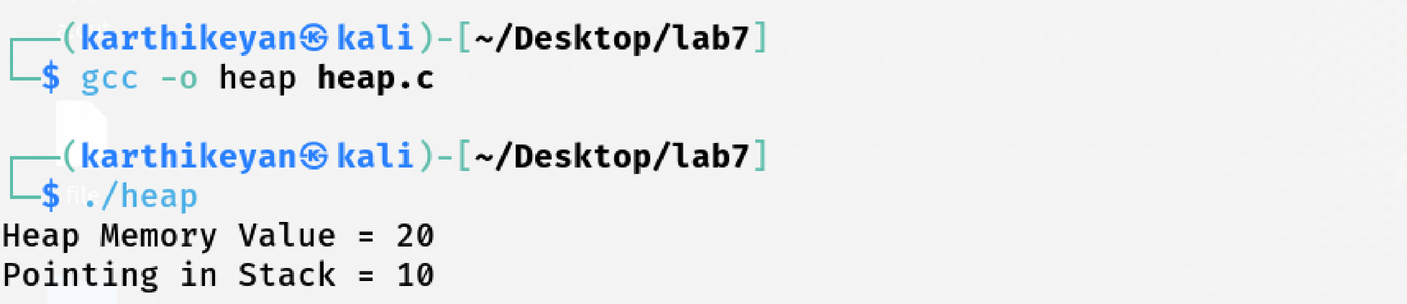
```
#include <stdio.h>
#include <string.h>

void copy(const char *argv) {
    char name[10];
    strncpy(name, argv, sizeof(name) - 1);
    name[sizeof(name) - 1] = '\0';
}

int main(int argc, char **argv) {
    if (argc > 1) {
        copy(argv[1]);
        printf("Exit\n");
    } else {
        printf("No argument provided\n");
    }
    return 0;
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc -o err err.c

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ./err 12345678901
Exit
```

Heap Memory Layout

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

int func() {
    int a = 10;
    int *aptr = &a;
    int *ptr = (int *)malloc(sizeof(int));
    *ptr = 20;
    printf("Heap Memory Value = %d\n", *ptr);
    printf("Pointing in Stack = %d\n", *aptr);
    free(ptr);
}

int main() {
    func();
    return 0;
}
```



```
┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ gcc -o heap heap.c

┌──(karthikeyan㉿kali)-[~/Desktop/lab7]
└─$ ./heap
Heap Memory Value = 20
Pointing in Stack = 10
```