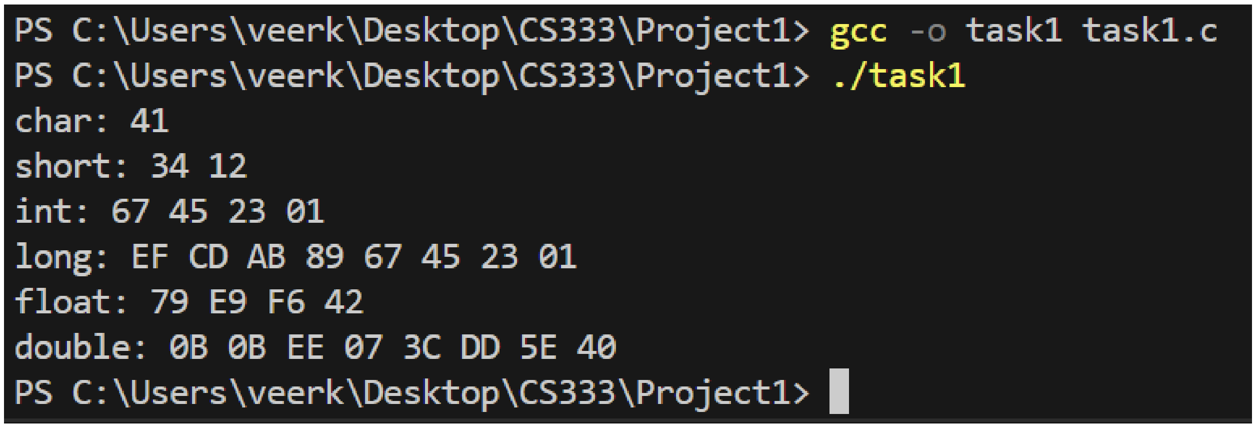
Veer Khosla

Website: https://sites.google.com/colby.edu/veers-cs333/home

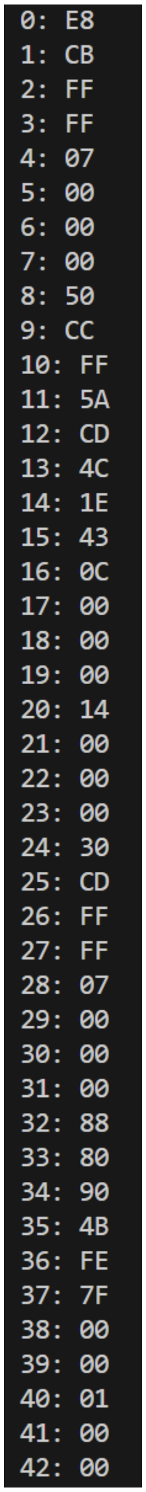
1. **Task 1**



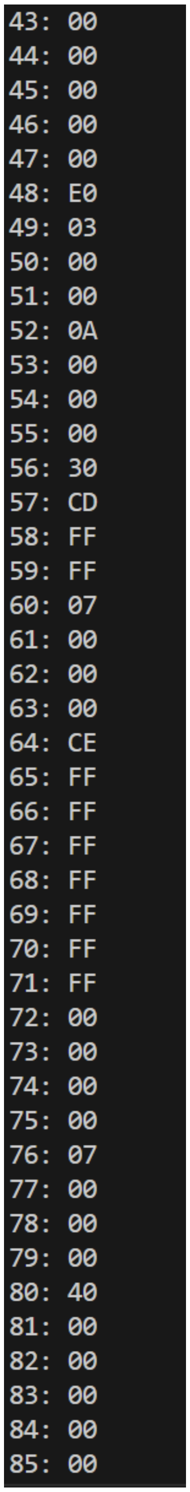
```
PS C:\Users\veerk\Desktop\CS333\Project1> gcc -o task1 task1.c
PS C:\Users\veerk\Desktop\CS333\Project1> ./task1
char: 41
short: 34 12
int: 67 45 23 01
long: EF CD AB 89 67 45 23 01
float: 79 E9 F6 42
double: 0B 0B EE 07 3C DD 5E 40
PS C:\Users\veerk\Desktop\CS333\Project1>
```

My machine is little-endian. This means the bytes are stored in memory in reverse order (same within each byte). This is characteristic of little-endian machines, prioritizing the most important byte at the highest memory address. Specifically, in the case of my short, 0x1234, we see 34 (most significant byte) before 12.

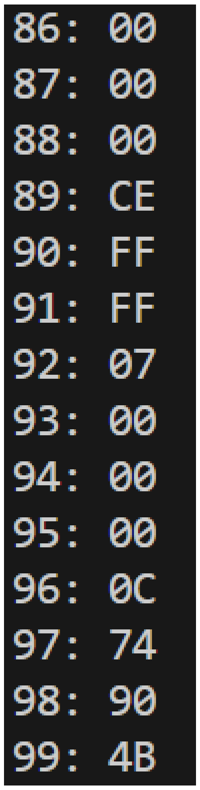
2. **Task 2**



```
0: E8
1: CB
2: FF
3: FF
4: 07
5: 00
6: 00
7: 00
8: 50
9: CC
10: FF
11: 5A
12: CD
13: 4C
14: 1E
15: 43
16: 0C
17: 00
18: 00
19: 00
20: 14
21: 00
22: 00
23: 00
24: 30
25: CD
26: FF
27: FF
28: 07
29: 00
30: 00
31: 00
32: 88
33: 80
34: 90
35: 4B
36: FE
37: 7F
38: 00
39: 00
40: 01
41: 00
42: 00
```



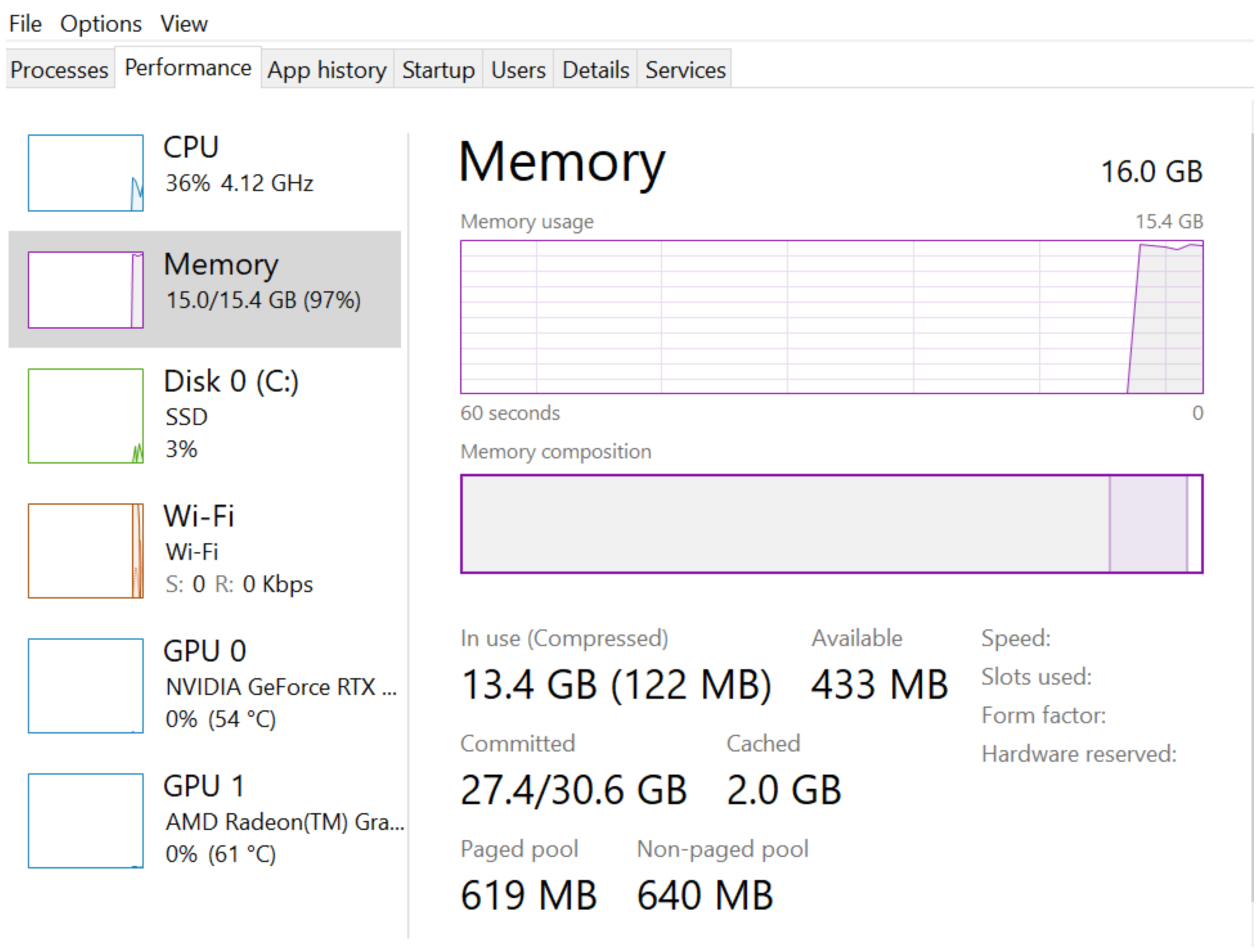
```
43: 00
44: 00
45: 00
46: 00
47: 00
48: E0
49: 03
50: 00
51: 00
52: 0A
53: 00
54: 00
55: 00
56: 30
57: CD
58: FF
59: FF
60: 07
61: 00
62: 00
63: 00
64: CE
65: FF
66: FF
67: FF
68: FF
69: FF
70: FF
71: FF
72: 00
73: 00
74: 00
75: 00
76: 07
77: 00
78: 00
79: 00
80: 40
81: 00
82: 00
83: 00
84: 00
85: 00
```



```
86: 00
87: 00
88: 00
89: CE
90: FF
91: FF
92: 07
93: 00
94: 00
95: 00
96: 0C
97: 74
98: 90
99: 4B
```

The variables that are initialized earlier are placed lower in the stack. We can see this with my int, 12, appearing at the 16th address (0C in hex). My char, Z, or 5A in hex, does not show up in the first 100 items. This makes sense as it is initialized later in the program. The other non-zero values in the first 100 items of the stack shown likely represents other important information from other parts of the program, such as the main function, or other important system information.

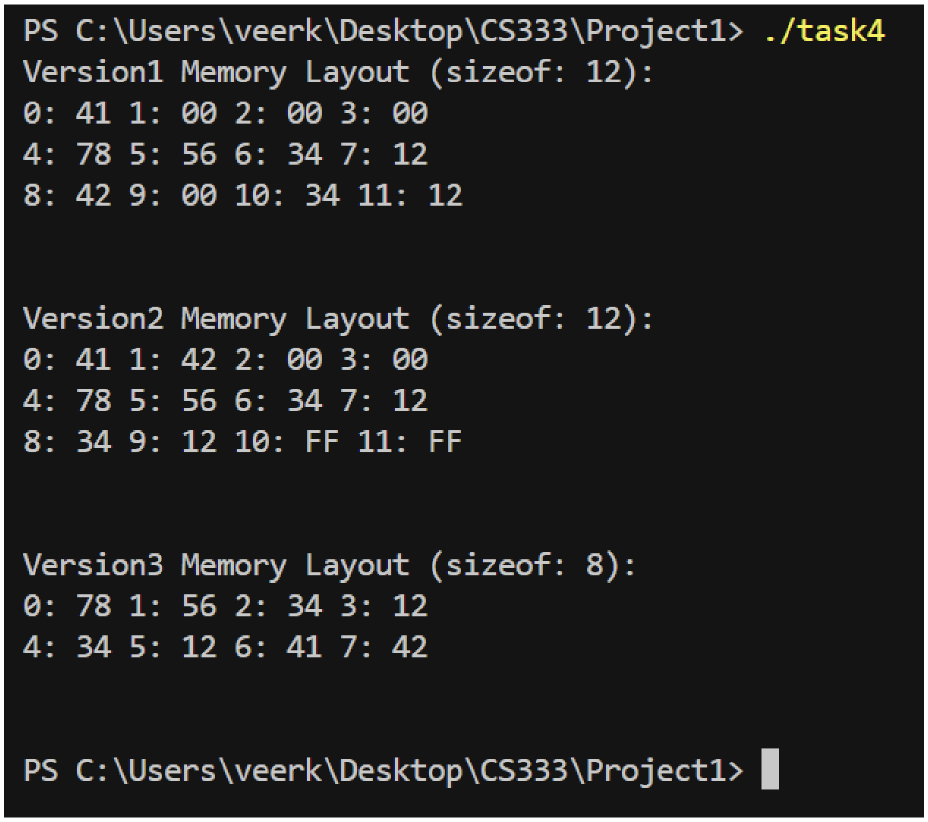
3. Task 3



In use (Compressed): 13.4 GB (122 MB)
Available: 433 MB
Speed:
Slots used:
Form factor:
Hardware reserved:
Committed: 27.4/30.6 GB
Cached: 2.0 GB
Paged pool: 619 MB
Non-paged pool: 640 MB

When not using the free statement, as shown above, memory quickly becomes allocated and becomes entirely occupied and used on the single program. However, with the free statement, memory is freed and released before iterating again, keeping memory usage low.

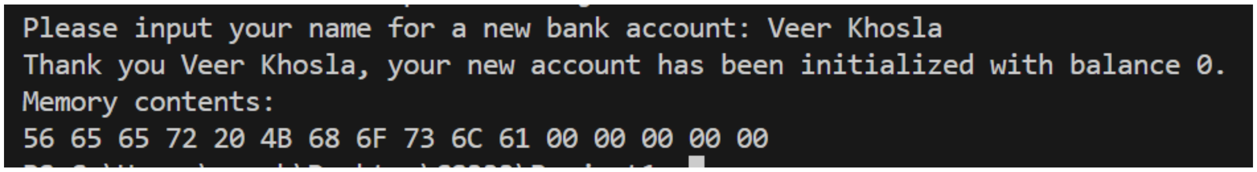
4. **Task 4**



```
PS C:\Users\veerk\Desktop\CS333\Project1> ./task4
Version1 Memory Layout (sizeof: 12):
0: 41 1: 00 2: 00 3: 00
4: 78 5: 56 6: 34 7: 12
8: 42 9: 00 10: 34 11: 12


Version2 Memory Layout (sizeof: 12):
0: 41 1: 42 2: 00 3: 00
4: 78 5: 56 6: 34 7: 12
8: 34 9: 12 10: FF 11: FF


Version3 Memory Layout (sizeof: 8):
0: 78 1: 56 2: 34 3: 12
4: 34 5: 12 6: 41 7: 42


PS C:\Users\veerk\Desktop\CS333\Project1>
```

The sizeof result does not match my expectation. This is likely due to the fact that the compiler inserting padding, as seen with the 00 bytes.

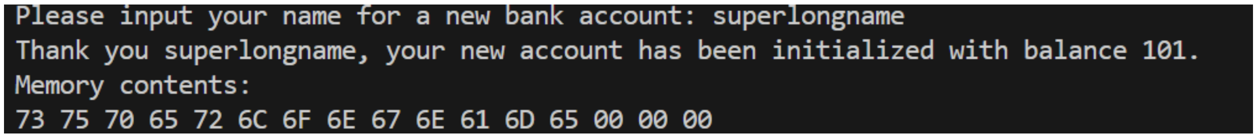
5. **Task 5**



```
Please input your name for a new bank account: Veer Khosla
Thank you Veer Khosla, your new account has been initialized with balance 0.
Memory contents:
56 65 65 72 20 4B 68 6F 73 6C 61 00 00 00 00 00
```

The above image is the output when inputting the legal string “Veer Khosla” for the name. There is no problem here as the balance is 0 and memory contents are accurate. On the other hand, The below image demonstrates the output when an illegal string “superlongname” is inputted.

Because I am exceeding the maximum name length (10), the remaining bytes are forced into the memory allocated for the balance.



```
Please input your name for a new bank account: superlongname
Thank you superlongname, your new account has been initialized with balance 101.
Memory contents:
73 75 70 65 72 6C 6F 6E 67 6E 61 6D 65 00 00 00
```