Task 1C: Memory Layout of Arrays using GDB

Objective:
To observe and understand the memory layout of different types of arrays in C using GDB:
- Single dimensional integer array
- Two dimensional integer array
- String array (2D character array)

This task demonstrates how arrays are stored in contiguous memory and how GDB can be used to inspect their addresses and values.

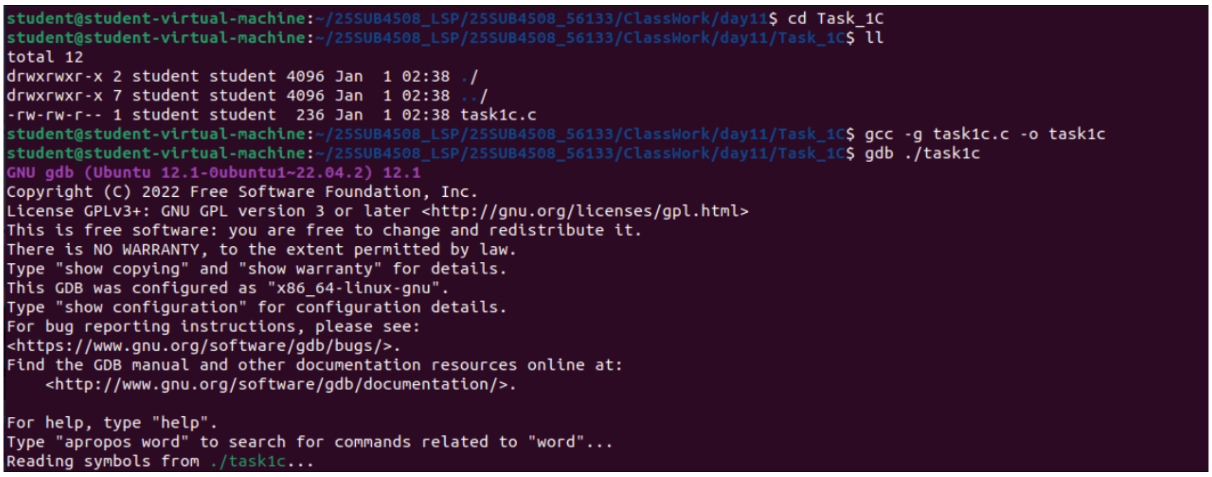
1. C Program Used

```
#include <stdio.h>

 int main() {
        int arr1D[5] = {10, 20, 30, 40, 50};
        int arr2D[2][3] = {{1, 2, 3}, {4, 5, 6}};
        char strArray[3][10] = {"ONE", "TWO", "THREE"};

        printf("GDB Memory Layout Observation\n");
        return 0;
 }
```

2. Compilation Steps

Compile the program with debug symbols enabled:

```
gcc -g task1c.c -o task1c
```

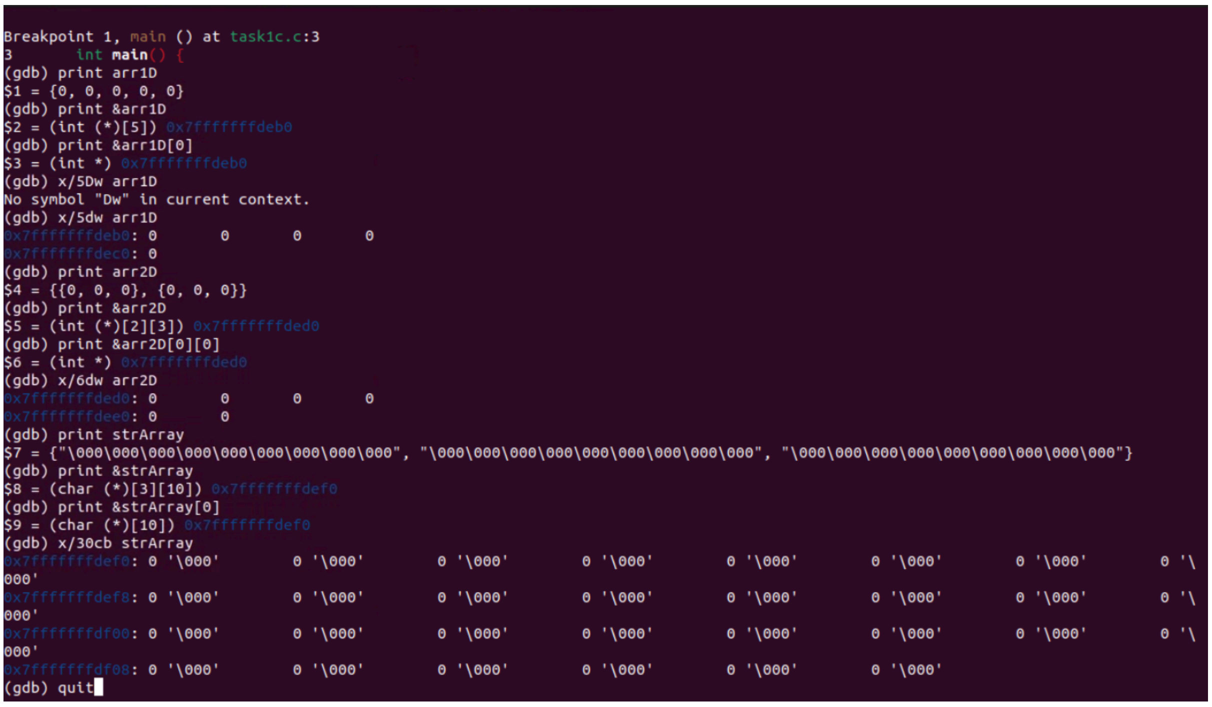


```
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/ClassWork/day11$ cd Task_1C
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/ClassWork/day11/Task_1C$ ll
total 12
drwxrwxr-x 2 student student 4096 Jan  1 02:38 ./
drwxrwxr-x 7 student student 4096 Jan  1 02:38 ../
-rw-rw-r-- 1 student student  236 Jan  1 02:38 task1c.c
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/ClassWork/day11/Task_1C$ gcc -g task1c.c -o task1c
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/ClassWork/day11/Task_1C$ gdb ./task1c
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./task1c...
```

3. GDB Commands Used

```
break main
run

print arr1D
print &arr1D
print &arr1D[0]
x/5dw arr1D

print arr2D
print &arr2D
print &arr2D[0][0]
x/6dw arr2D

print strArray
print &strArray
print &strArray[0]
x/30cb strArray
```

4. GDB Outputs



```
Breakpoint 1, main () at task1c.c:3
3       int main() {
(gdb) print arr1D
$1 = {0, 0, 0, 0, 0}
(gdb) print &arr1D
$2 = (int (*)[5]) 0x7fffffffdeb0
(gdb) print &arr1D[0]
$3 = (int *) 0x7fffffffdeb0
(gdb) x/5Dw arr1D
No symbol "Dw" in current context.
(gdb) x/5dw arr1D
0x7fffffffdeb0: 0       0       0       0
0x7fffffffdec0: 0
(gdb) print arr2D
$4 = {{0, 0, 0}, {0, 0, 0}}
(gdb) print &arr2D
$5 = (int (*)[2][3]) 0x7fffffffded0
(gdb) print &arr2D[0][0]
$6 = (int *) 0x7fffffffded0
(gdb) x/6dw arr2D
0x7fffffffded0: 0       0       0       0
0x7fffffffdee0: 0       0
(gdb) print strArray
$7 = {"\000\000\000\000\000\000\000\000\000", "\000\000\000\000\000\000\000\000\000", "\000\000\000\000\000\000\000\000\000"}
(gdb) print &strArray
$8 = (char (*)[3][10]) 0x7fffffffdef0
(gdb) print &strArray[0]
$9 = (char (*)[10]) 0x7fffffffdef0
(gdb) x/30cb strArray
0x7fffffffdef0: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\
000'
0x7fffffffdef8: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\
000'
0x7fffffffdf00: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\
000'
0x7fffffffdf08: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'
(gdb) quit
```

5. Observations

- All arrays are stored in contiguous memory locations.
- 2D arrays are stored row-wise (row-major order).
- String arrays reserve fixed memory per string.
- Address arithmetic follows data type size.

6. Conclusion

This task helps understand how arrays are stored in memory and how GDB can be used to inspect memory layouts. Such knowledge is essential for debugging and pointer manipulation.