SEMESTER 3 - PROJECT
CROSSWORD PUZZLE SOLVER
PHASE 2 - PROJECT REPORT

E/19/057 - COLOMBAGE C.O.

E/19/409 - UDUGAMASOORIYA D.P.

https://github.com/CColombus/SEM3_Crossword

INTRODUCTION

The team developed a puzzle solver that employed static memory allocation techniques in the first phase of the project. In this phase (phase 2), we improved the solver using dynamic memory allocation techniques in contrast to the conventional static memory allocation method.

OBJECTIVE

To use dynamic memory allocation techniques for more efficient memory resources and improve the overall performance of the puzzle solver.

PROGRAMMING TOOLS

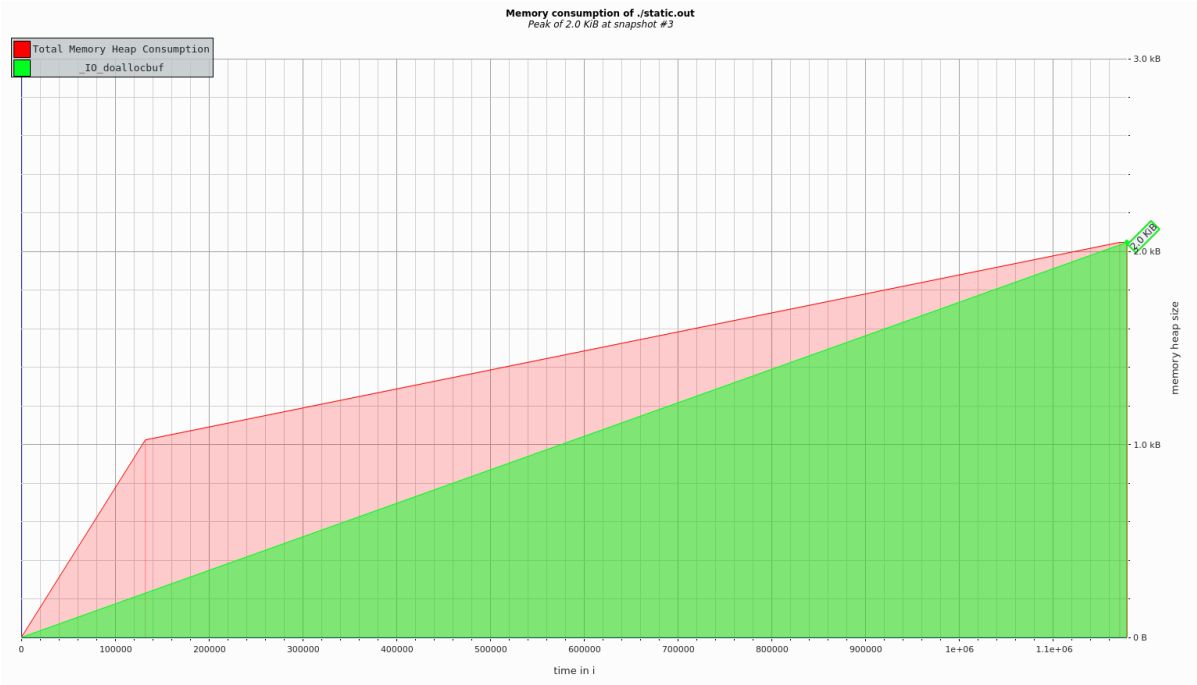
1. C programming language
2. Massif-visualizer (heap profiler)
3. Bash

PROCEDURE

1. Identify the variables and data structures in the program that are currently using static memory allocation.
2. Replace these statically allocated variables and data structures with dynamically allocated equivalents.
3. Implement functions or methods for allocating and deallocating memory dynamically, such as malloc() and free().
4. Carefully manage the dynamically allocated memory throughout the program, including proper initialization, error checking, and releasing of memory when it is no longer needed.
5. Test the program thoroughly to ensure no memory leaks or other issues related to dynamic memory management using the “Massif heap profiler” and “memusage” bash script.

DATA

While the program uses static memory allocation for comparison.



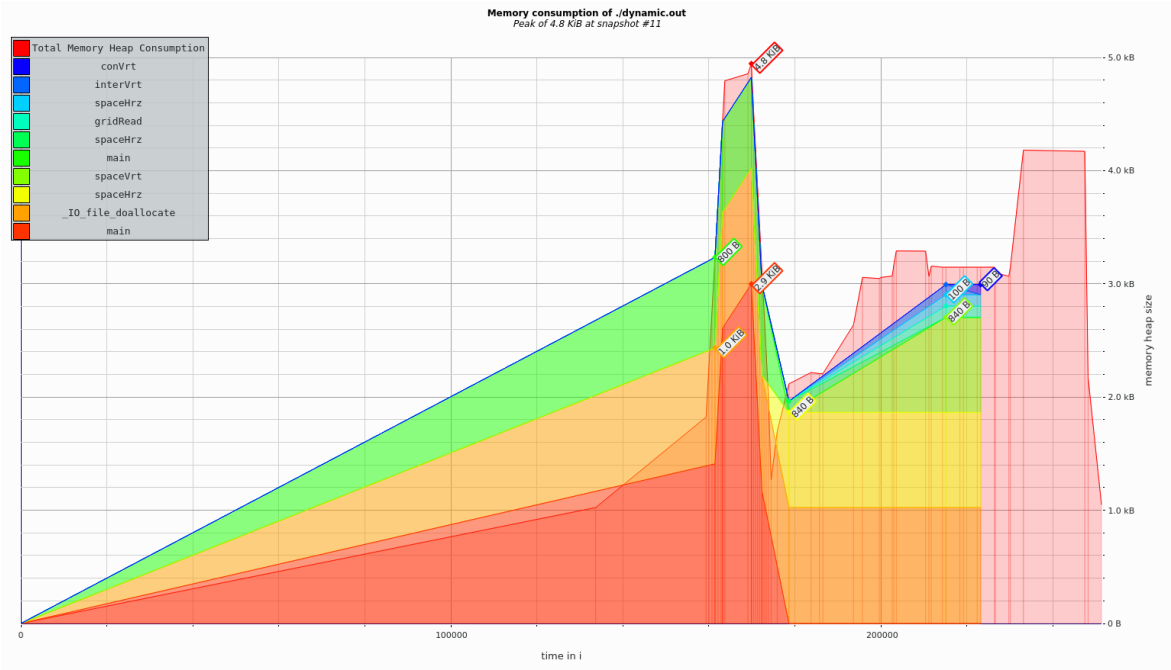
There was no discernable difference in memory usage regarding different test cases for static memory allocation.

CASE 1

```
*******#***
**#######**
*******#***
*******#***
***#######*
*******#*#*
*******#*#*
********#**
********#**
********#**

ICELAND
MEXICO
PANAMA
```

ALMATY

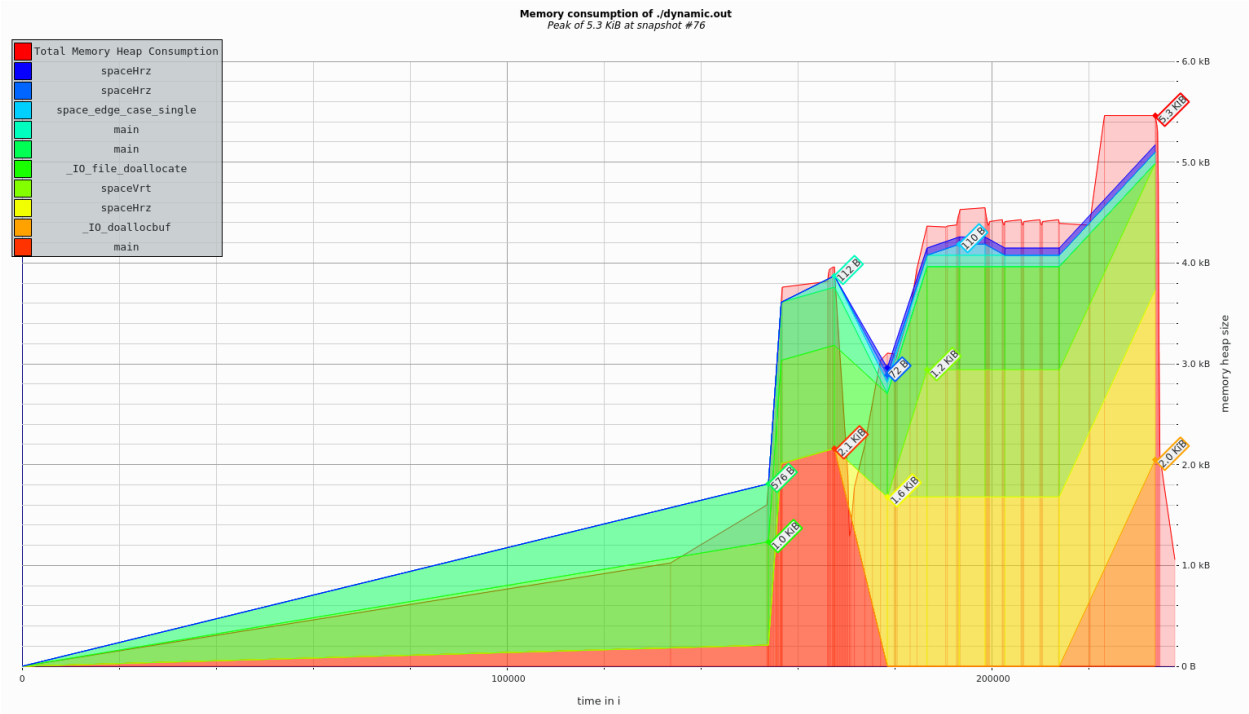


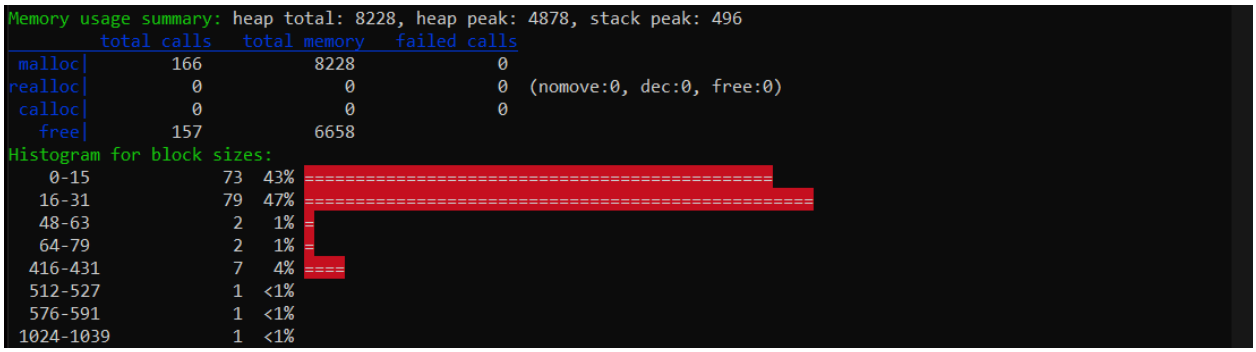

```
Memory usage summary: heap total: 8313, heap peak: 4433, stack peak: 528
         total calls   total memory   failed calls
 malloc|         211           8313              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|         205           6752
Histogram for block sizes:
    0-15            96  45% ============================================
   16-31           104  49% ================================================
   32-47             2  <1%
   96-111            2  <1%
  416-431            4   1% =
  512-527            1  <1%
  800-815            1  <1%
 1024-1039           1  <1%
```

CASE 2

```
********###
***#****#**
*#*#****#**
#########*
*#*#****#**
*#########
*#########
*#########
*#########

Beholder
sTOrm
tiger
two
neEDlE
evE
DOME
```

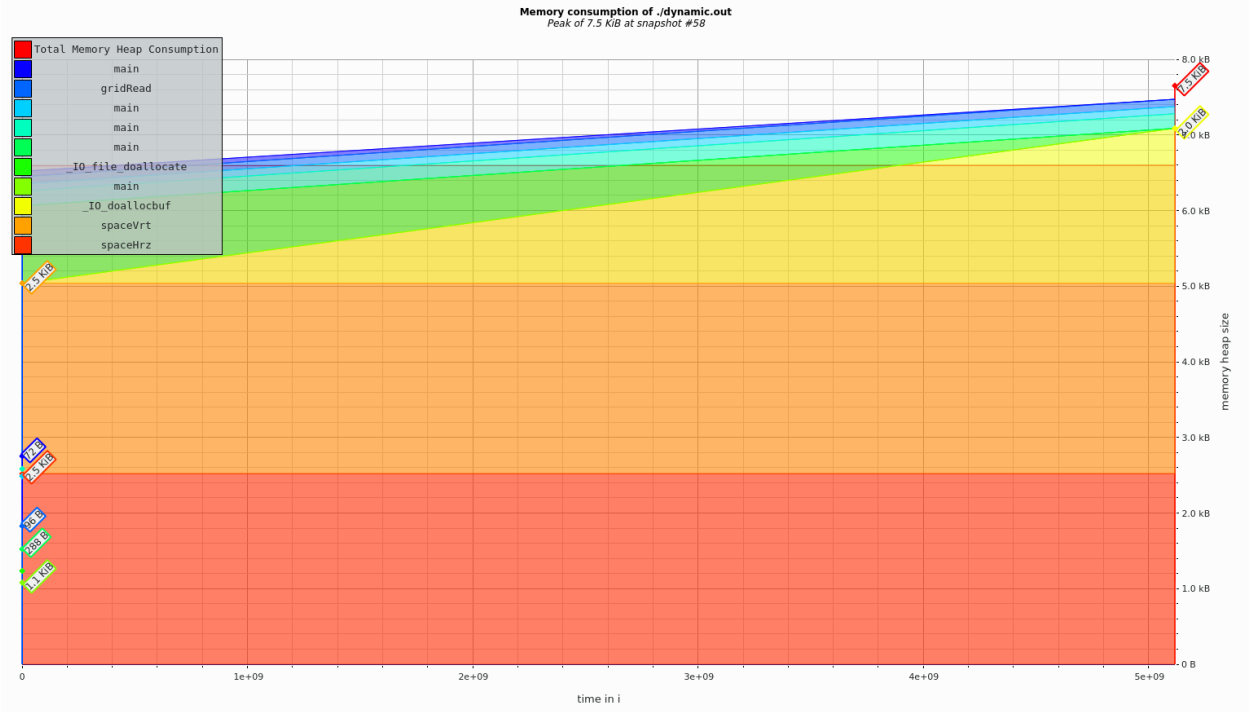




CASE 3

```
##########
#*#*#*****#
#####*#####
#*#*#*****#
#*#*#######
**#******#**
########*#*#
#****#*#*#
#####*#####
#****#*#*#
###*#######

EXAMPLE
YEA
OWING
TAPED
SURFEIT
JOYLESS
LITHE
ORBIT
YEW
TERSELY
EVOKE
ALIMONY
PAGES
ENTER
AUDIT
JOLLY
EVENT
SCOUR
EQUABLE
DITZY
```

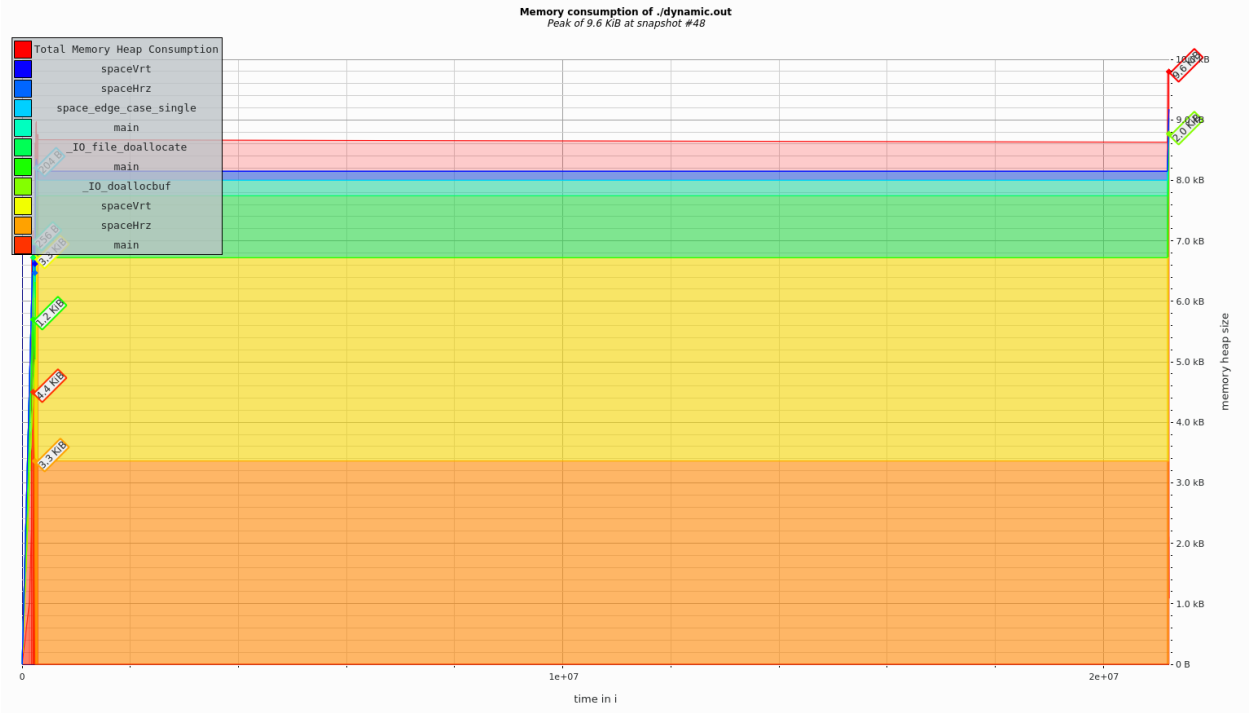



```
Memory usage summary: heap total: 8824, heap peak: 7140, stack peak: 496
         total calls   total memory   failed calls
 malloc|         135           8824              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|         121           7216
Histogram for block sizes:
    0-15            68  50% ==================================================
   16-31            48  35% ====================================
   32-47             2   1% =
   96-111            2   1% =
  288-303            1  <1%
  416-431           12   8% ========
  512-527            1  <1%
 1024-1039           1  <1%
```

CASE 4

```
***#******
***#******
**###*****
##########
#*********
##########
**########
**#*******
**########
**#*#*****
*********#
*********
***##*****
****#*****

cook
not
no
babara
kanishka
bat
dinali
book
sat
tap
aba
ir
piyumika
al
yoyo
moda
```

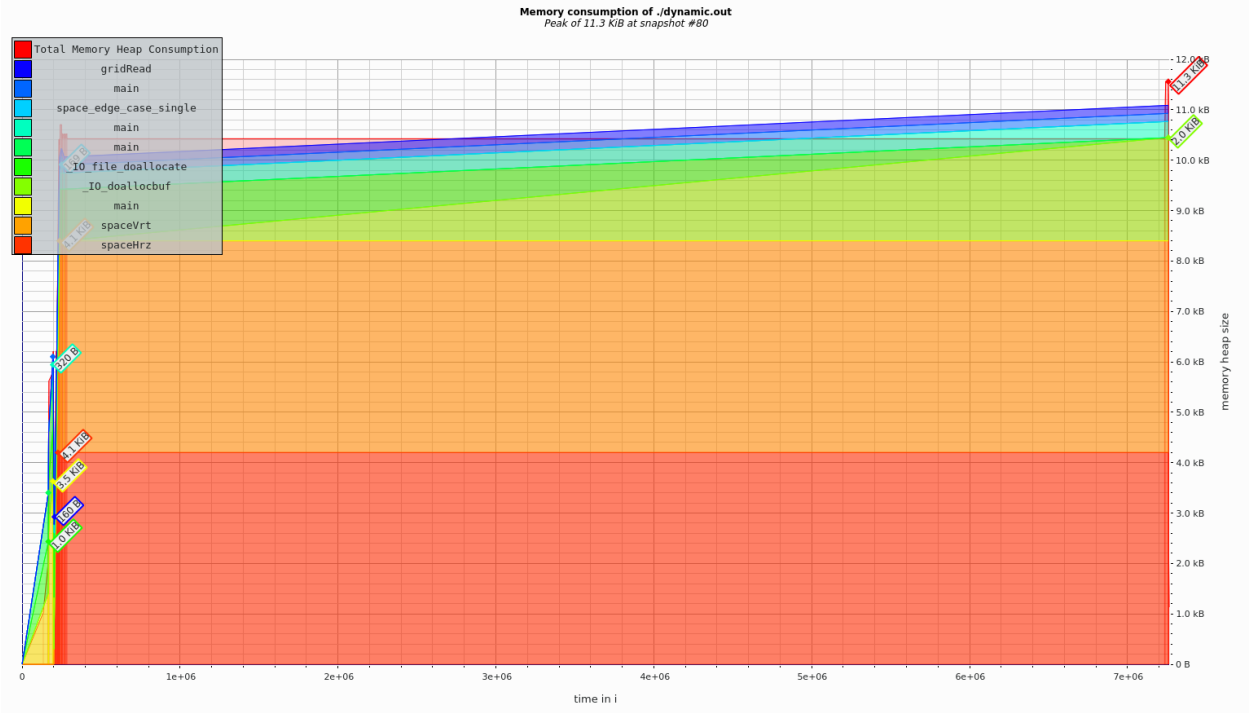



```
Memory usage summary: heap total: 16087, heap peak: 9133, stack peak: 496
        total calls   total memory   failed calls
 malloc|         327          16087              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|         309          14486
Histogram for block sizes:
    0-15           138  42% ========================================
   16-31           166  50% ================================================
  128-143            2  <1%
  144-159            2  <1%
  416-431           16   4% ====
  512-527            1  <1%
 1024-1039           1  <1%
 1200-1215           1  <1%
```

CASE 5

```
##########*###
#*#*#*#####*#
#####*#####
#*#*#*#####*#
#*#*#######
**########**
########*#*#
#****#*#*#*#
#####*#####
#****#*#*#*#
###*#######

EXAMPLE
YEA
OWING
TAPED
SURFEIT
JOYLESS
LITHE
ORBIT
YEW
TERSELY
EVOKE
ALIMONY
PAGES
ENTER
AUDIT
JOLLY
EVENT
SCOUR
EQUABLE
DITZY
```




```
Memory usage summary: heap total: 16661, heap peak: 11047, stack peak: 544
         total calls   total memory   failed calls
 malloc|         289          16661              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|         267          15017
Histogram for block sizes:
    0-15           121  41% ========================================
   16-31           141  48% ================================================
  112-127            2  <1%
  160-175            2  <1%
  416-431           20   6% =======
  512-527            1  <1%
  960-975            1  <1%
 1024-1039           1  <1%
```

RESULTS

AVERAGE EXECUTION TIME (in mili-seconds)

TEST CASE	STATIC MEMORY ALLOCATION	DYNAMIC MEMORY ALLOCATION
CASE 1	1	1
CASE 2	1	2
CASE 3	413	448
CASE 4	4	4
CASE 5	3	2
AVERAGE TIME	84.4	91.4

MEMORY ALLOCATION

TEST CASE	STATIC MEMORY ALLOCATION	DYNAMIC MEMORY ALLOCATION
CASE 1	2.0 KiB	4.8 KiB
CASE 2		5.3 KiB
CASE 3		7.5 KiB
CASE 4		9.6 KiB
CASE 5		11.3 KiB

CONCLUSION

According to the above statistics, we can observe that the dynamically memory allocated program consumed more memory and time than the statically memory allocated program.

Although it’s not necessarily true that a dynamically allocated program will consume more memory than a statically allocated one, in our program, it may have been because

1. Over allocation: When dynamically allocating memory, it's possible that we may have over-allocated memory. This can lead to wasted memory, which can cause the program to consume more memory than it needs to.
2. Pointers: When using dynamic memory allocation, we have to use pointers to reference the memory that has been allocated. These pointers take up memory themselves, which can cause the program to consume more memory than it needs to.

That being said, the main advantage of dynamic memory allocation is that the program can use only the memory it needs, so it's more efficient than static memory allocation. For example, the version that uses dynamic memory allocation can allocate memory for arrays only after the user inputs the puzzle grid; this way, the program can adapt to the user's needs and use only the memory it needs.