Group 31

PROJECT REPORT

CO222 PROGRAMMING METHODOLOGY

GROUP 31

E/19/129 K.H. GUNAWARDANA

E/19/408 S.J. UBAYASIRI

Introduction

In this project, we were supposed to write a program to solve a grid using a given list of words.

Specifications

The inputs for the program are the grid and the list of words.

The valid inputs for the grid are '#' (indicate spaces), '*' (indicate blocks) and letters. If there are any other inputs given, the program will indicate the inputs are INVALID and exit.

The grid is considered to be solvable if and only if all spaces in the grid can be filled using all the words given. Otherwise, the program will indicate the puzzle is IMPOSSIBLE to solve.

Algorithm and Implementation

The grid is being checked for solvability by a recursive algorithm.

First, the lengths and coordinates of all the spaces and word lengths are recorded.

Then, the word array is sorted according to the number of words with the same length in order to obtain the minimum spanning tree.

Then, the puzzle is filled by a recursive function. At every node, a copy of the current instances of the variables is recorded in order to backtrack if necessary.

Project Repository

Includes the source codes of static and dynamic allocation and a simple test-bench created using make.

GitHub Repo - https://github.com/KATTA-00/CO222_Project.git

The cases included in the report can be tested using following test bench.

Test bench - https://github.com/KATTA-00/CO222_Project/tree/master/test

Static Allocation of Memory

```c
// define the macros
#define gridRowFix 20
#define gridColFix 20
#define wordsNum 30
#define wordLen 10
#define maxSpacelen 30
#define maxSpace 20
```

```c
/*
    define the global variables
        grid - store the grid
        words - store the words
        wordLens - store the length of the words
        wordCount - word count
        spacesCords - store the spaces coordinates together
        spaceLens - legths arr of spaces
        spaceCount - count of spaces

*/
char grid[gridRowFix][gridColFix];
char words[wordsNum][wordLen];
int wordLens[wordsNum];
int wordCount = 0;
coordinate spacesCords[maxSpacelen][maxSpace];
int spaceLens[maxSpacelen] = {0};
int spaceCount = 0;
int flag = 0;
// initalize row and col
int gridRow = gridColFix;
int gridCol = gridColFix;
```

Figure 1: Static allocation of arrays

All arrays are allocated with a fixed length in static allocation.

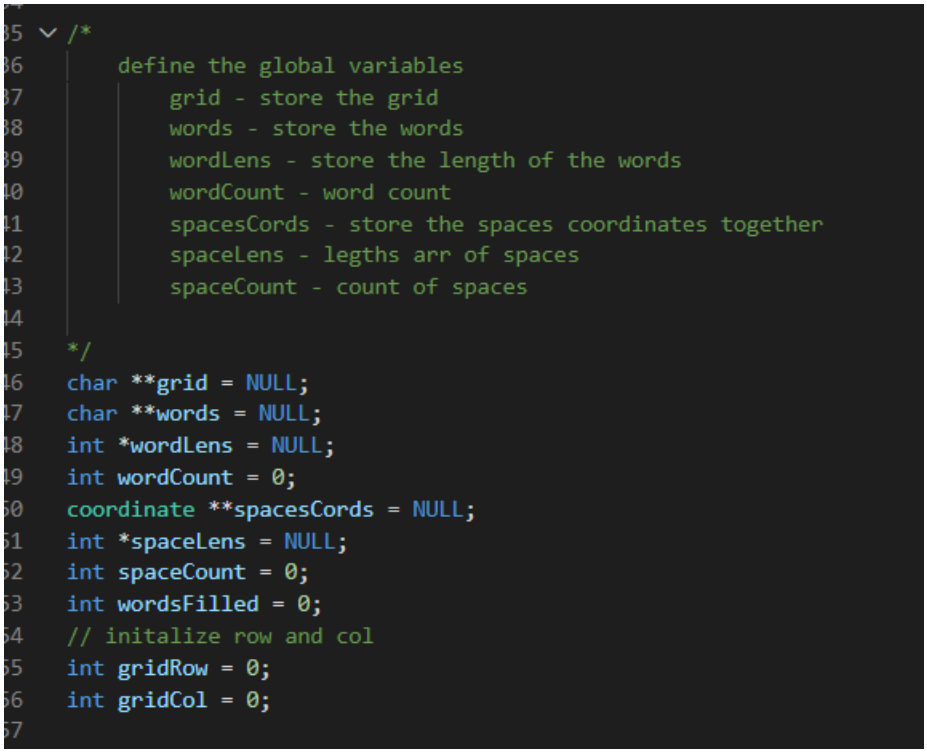
Advantages: Less execution time

Memory is automatically freed as the function finishes executing

Drawbacks: A memory wastage can occur in smaller grids

Larger grids (grid size > allocated array size) cannot be solved using static allocation

Dynamic Allocation of Memory



```
/*
    define the global variables
        grid - store the grid
        words - store the words
        wordLens - store the length of the words
        wordCount - word count
        spacesCords - store the spaces coordinates together
        spaceLens - legths arr of spaces
        spaceCount - count of spaces

*/
char **grid = NULL;
char **words = NULL;
int *wordLens = NULL;
int wordCount = 0;
coordinate **spacesCords = NULL;
int *spaceLens = NULL;
int spaceCount = 0;
int wordsFilled = 0;
// initalize row and col
int gridRow = 0;
int gridCol = 0;
```

Figure 2: Pointers for dynamically allocating of arrays

Arrays that require an altering length specific to a given case are dynamically allocated.

Unlike static allocation, grids with any length can be filled using dynamic allocation.

Comparison

Test Case 1

Static Allocation

```
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222
ethodology/CO222_Project/PuzzleStatic$ ./puzzle-static
****
####
****
*###

FIRE
CAT

****
FIRE
****
```

Output of the code

```
Ouput >>>
****
FIRE
****
*CAT
Time - 0.000173(seconds)
```

Execution time

```
CO222_Project/PuzzleStatic$ memusage ./puzzle-static
****
####
****
*###

FIRE
CAT

****
FIRE
****
*CAT

Memory usage summary: heap total: 2048, heap peak: 2048, stack peak: 176
         total calls   total memory   failed calls
 malloc|           2           2048              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|           0              0
Histogram for block sizes:
 1024-1039           2 100% ==========================================
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodology/
```

Total memory usage by the program.

Figure 3: Output and memory usage of test case 1(Static allocation)

Dynamic Allocation

```
CO222_Project/PuzzleDynamic$ ./puzzle-dynamic
****
####
****
*###

FIRE
CAT

****
FIRE
****
*CAT
```

Output of the code

```
Ouput >>>
****
FIRE
****
*CAT
Time - 0.000261(seconds)
```

Execution time

.

```
==472== Memcheck, a memory error detector
==472== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==472== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==472== Command: ./puzzle-dynamic
==472==
****
####
****
*###

FIRE
CAT

****
FIRE
****
*CAT
==472==
==472== HEAP SUMMARY:
==472==     in use at exit: 0 bytes in 0 blocks
==472==   total heap usage: 65 allocs, 65 frees, 11,117 bytes allocated
==472==
==472== All heap blocks were freed -- no leaks are possible
==472==
==472== For lists of detected and suppressed errors, rerun with: -s
==472== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

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

FIRE
CAT

****
FIRE
****
*CAT

Memory usage summary: heap total: 10664, heap peak: 2237, stack peak: 512
         total calls   total memory   failed calls
 malloc|          15          10400              0
realloc|          50            264              0  (nomove:31, dec:15, free:0)
 calloc|           0              0              0
   free|          27           2453
Histogram for block sizes:
    0-15           26  40% ==========================================================
   16-31           20  30% ============================================
   32-47            9  13% ====================
 1024-1039          10  15% ======================
```

Total memory usage by the program.

Figure 4: Output and memory usage of test case 1(Dynamic allocation)

Test Case 2

Static Allocation

```
CO222_Project/PuzzleStatic$ ./puzzle-static
******#***
**######**
******#***
******#***
***######*
******#*#*
******#*#*
********#*
********#*
********#*

ICELAND
MEXICO
PANAMA
ALMATY

******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodology
```

Output of the code

```
Ouput >>>
******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
Time - 0.000543(seconds)
```

Execution time

```
CO222_Project/PuzzleStatic$ memusage ./puzzle-static
******#***
**######**
******#***
******#***
***######*
******#*#*
******#*#*
********#*
********#*
********#*

ICELAND
MEXICO
PANAMA
ALMATY

******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*

Memory usage summary: heap total: 2048, heap peak: 2048, stack peak: 176
         total calls   total memory   failed calls
 malloc|           2           2048              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|           0              0
Histogram for block sizes:
 1024-1039           2 100% ==========================================
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodology/
```

Total memory usage by the program.

Figure 5: Output and memory usage of test case 2(Static allocation)

Dynamic Allocation

```
CO222_Project/PuzzleDynamic$ ./puzzle-dynamic
******#***
**#######**
******#***
******#***
***#######*
******#*#*
******#*#*
********#*
********#*
********#*

ICELAND
MEXICO
PANAMA
ALMATY

******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodol
```

Output of the code

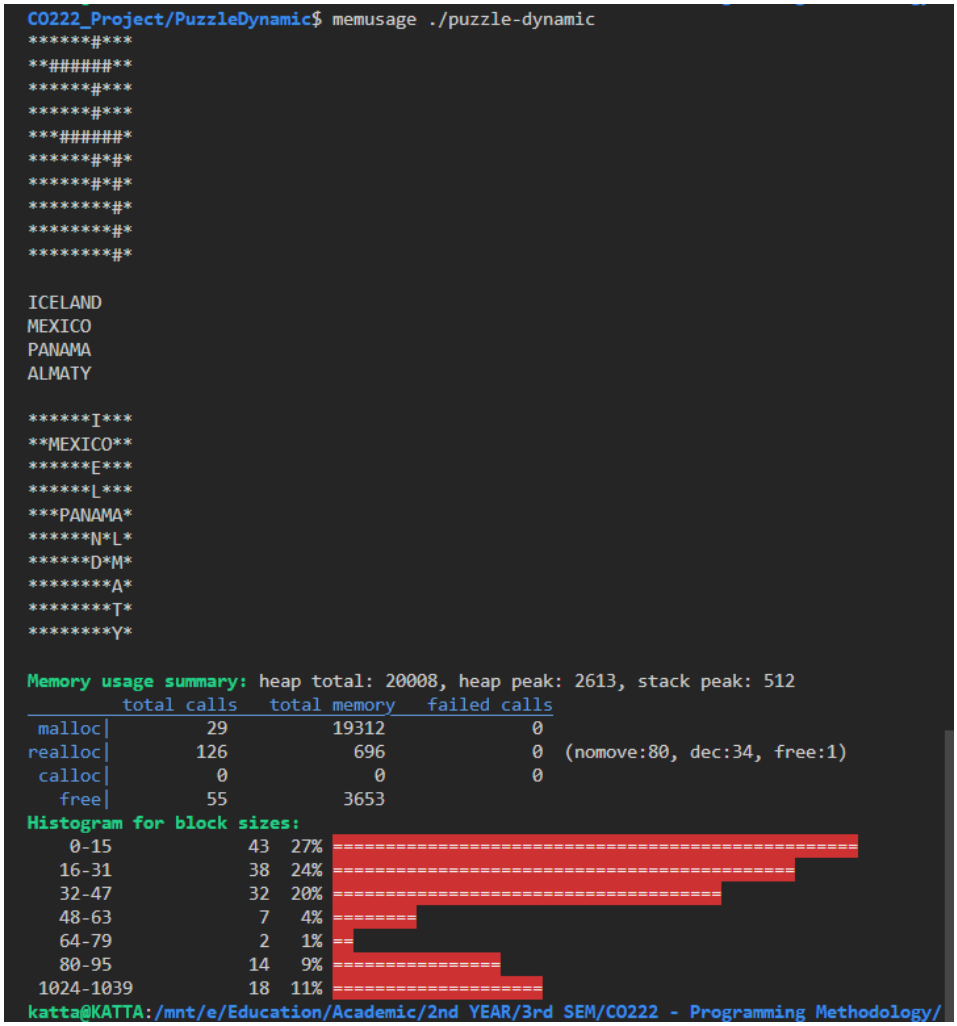
```
Ouput >>>
******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
Time - 0.000804(seconds)
```

Execution time

```
CO222_Project/PuzzleDynamic$ valgrind ./puzzle-dynamic
==535== Memcheck, a memory error detector
==535== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==535== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==535== Command: ./puzzle-dynamic
==535==
******#***
**#######**
******#***
******#***
***#######*
******#*#*
******#*#*
********#*
********#*
********#*

ICELAND
MEXICO
PANAMA
ALMATY

******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
==535==
==535== HEAP SUMMARY:
==535==     in use at exit: 0 bytes in 0 blocks
==535==   total heap usage: 154 allocs, 154 frees, 22,317 bytes allocated
==535==
==535== All heap blocks were freed -- no leaks are possible
==535==
==535== For lists of detected and suppressed errors, rerun with: -s
==535== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Meth
```



Total memory usage by the program.

Figure 6: Output and memory usage of test case 2(Dynamic allocation)

Test Case 3

Static Allocation

```
CO222_Project/PuzzleStatic$ ./puzzle-static
#####*####
#**#######*
#####*####
#*######*#*
###***#####
########***###
*#*######*#
####*#####
*#######**#
####*#####

MELON
MESA
REGIME
MIMOSA
MIDAS
TUNA
ALTAR
SIR
EROS
SENIOR
ANTS
RUPEE
ART
STREAM
SERUM
PEON
NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS
```

```
PEON
NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS

MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/C
```

Output of the code

```
Ouput >>>
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time - 0.001524(seconds)
```

Execution time

```
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM

Memory usage summary: heap total: 2048, heap peak: 2048, stack peak: 176
         total calls   total memory   failed calls
 malloc|           2           2048              0
realloc|           0              0              0  (nomove:0, dec:0, free:0)
 calloc|           0              0              0
   free|           0              0
Histogram for block sizes:
 1024-1039           2 100% ==============================================
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodology/
```

Total memory usage by the program.

Figure 7: Output and memory usage of test case 3(Static allocation)

Dynamic Allocation

```
CO222_Project/PuzzleDynamic$ ./puzzle-dynamic
#####*#####
#**#######*
#####*#####
#*######*#*
###***#####
#######*###
*#*######*#
####*#####
*########*#
####*#####

MELON
MESA
REGIME
MIMOSA
MIDAS
TUNA
ALTAR
SIR
EROS
SENIOR
ANTS
RUPEE
ART
STREAM
SERUM
PEON
NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS
```

```
EROS
SENIOR
ANTS
RUPEE
ART
STREAM
SERUM
PEON
NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS

MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/
```

Output of the code

```
Ouput >>>
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time - 0.001656(seconds)
```

```
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
==842==
==842== HEAP SUMMARY:
==842==     in use at exit: 0 bytes in 0 blocks
==842==   total heap usage: 425 allocs, 425 frees, 67,866 bytes allocated
==842==
==842== All heap blocks were freed -- no leaks are possible
==842==
==842== For lists of detected and suppressed errors, rerun with: -s
==842== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Meth
```

Execution time

```
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM

Memory usage summary: heap total: 48160, heap peak: 4522, stack peak: 512
         total calls   total memory   failed calls
 malloc|          55          45936              0
realloc|         370           2224              0  (nomove:243, dec:62, free:0)
 calloc|           0              0              0
   free|         108           5562
Histogram for block sizes:
    0-15            85  20% ==========================================
   16-31            92  21% ==============================================
   32-47            54  12% ============================
   48-63            20   4% ==========
   64-79            19   4% ==========
   80-95            29   6% ===============
   96-111           18   4% =========
  112-127           12   2% ======
  128-143            4  <1% ==
  144-159            4  <1% ==
  160-175            5   1% ==
  176-191            7   1% ===
  192-207            7   1% ===
  208-223            6   1% ===
  224-239           10   2% =====
  240-255            9   2% ====
 1024-1039          44  10% ======================
katta@KATTA:/mnt/e/Education/Academic/2nd YEAR/3rd SEM/CO222 - Programming Methodology/
```

Total memory usage by the program.

Figure 8: Output and memory usage of test case 3(Dynamic allocation)

Conclusion

- Run-time of dynamically allocated program was higher than the statically allocated program. This might be due to the high number of function calls (malloc() and realloc() functions).
- Also allocating memory in heap is slower than allocating memory in stack. Thus, dynamic allocation is slower than static allocation.
- Memory wastage is less in dynamic allocation than in static allocation.
- No memory leakages were occurred because all arrays were freed.
- Total heap and heap peak are variating with different test cases in dynamic allocation whereas stack peak is a constant for all test cases in static allocation.