Crossword Puzzle Solver

K. H. J. D. Premachandra - E/19/295

L. M. A. H. Premawansha - E/19/300

17.01.2023

Introduction

In this project, a C program is written to solve a crossword puzzle. The program takes inputs from the standard input, a puzzle grid, and a set of words and then prints the solved puzzle to the standard output. In the project, two main memory allocation methods are used to implement the code. Those are static memory allocation and dynamic memory allocation. Here in the following paragraphs, the major differences between the two methods are discussed.

Memory space usage differences

Static memory allocation in C refers to the assignment of memory to a variable at the time of compilation, whereas dynamic memory allocation refers to the assignment of memory to a variable during runtime.

Because the memory is only allocated once at build time and the necessary quantity of memory is known in advance, static memory allocation is more memory-efficient. This lowers the possibility of memory leaks and makes it simpler to optimize the use of memory. The size of the RAM allotted, however, is fixed and cannot be altered while the program is running. This indicates that the application will need to be recompiled with a larger memory allocation if it requires more memory than what was allocated at compilation time.

Memory usage for static case

```
e19300@aiken:~/CO222/MiniProject$ valgrind ./puzzel-static
==64138== Memcheck, a memory error detector
==64138== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==64138== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==64138== Command: ./puzzel-static
==64138==
*#**
####
*#**
****

FLY
GLUE

*F**
GLUE
*Y**
****
Execution time is: 0.041290
==64138==
==64138== HEAP SUMMARY:
==64138==     in use at exit: 0 bytes in 0 blocks
==64138==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==64138==
==64138== All heap blocks were freed -- no leaks are possible
==64138==
==64138== For counts of detected and suppressed errors, rerun with: -v
==64138== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
e19300@aiken:~/CO222/MiniProject$
```

On the other hand, dynamic memory allocation enables memory allocation and deallocation during runtime. This means that the software can change how much memory it consumes based on what is necessary. This is advantageous for applications that need to handle vast amounts of data or data with unknown sizes. Dynamic memory allocation, however, can result in memory fragmentation and memory leaks if improperly handled because the quantity of memory required is not known in advance.

Memory usage for dynamic case

```
e19300@aiken:~/CO222/MiniProject$ clear
e19300@aiken:~/CO222/MiniProject$ valgrind ./puzzel-dynamic
==64302== Memcheck, a memory error detector
==64302== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==64302== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==64302== Command: ./puzzel-dynamic
==64302==
*#**
####
*#**
****

FLY
GLUE==64302== Conditional jump or move depends on uninitialised value(s)
==64302==    at 0x108C00: countRows_P (in /home/e19300/CO222/MiniProject/puzzel-dynamic)
==64302==    by 0x108DBD: main (in /home/e19300/CO222/MiniProject/puzzel-dynamic)
==64302==
*#**
####
*#**
****

FLY
GLUE

^Z==64302== Conditional jump or move depends on uninitialised value(s)
==64302==    at 0x108C3F: countRows_W (in /home/e19300/CO222/MiniProject/puzzel-dynamic)
==64302==    by 0x108E80: main (in /home/e19300/CO222/MiniProject/puzzel-dynamic)
==64302==
IMPOSSIBLE
==64302==
==64302== HEAP SUMMARY:
==64302==     in use at exit: 135,168 bytes in 514 blocks
==64302==   total heap usage: 516 allocs, 2 frees, 137,216 bytes allocated
==64302==
==64302== LEAK SUMMARY:
==64302==    definitely lost: 0 bytes in 0 blocks
==64302==    indirectly lost: 0 bytes in 0 blocks
==64302==      possibly lost: 0 bytes in 0 blocks
==64302==    still reachable: 135,168 bytes in 514 blocks
==64302==         suppressed: 0 bytes in 0 blocks
==64302== Rerun with --leak-check=full to see details of leaked memory
==64302==
==64302== For counts of detected and suppressed errors, rerun with: -v
==64302== Use --track-origins=yes to see where uninitialised values come from
==64302== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
e19300@aiken:~/CO222/MiniProject$
```

Overall, there are benefits and drawbacks to both dynamic and static memory allocation. When it is considered which form of memory allocation is more appropriate based on the needs of the software, When memory requirements are

unknown in advance or change as a program is being run, dynamic memory allocation can be helpful. When memory size is predetermined and stays constant throughout program execution, static memory allocation is advantageous. This enables quicker execution time and improved memory management and optimization.

Static memory allocation is generally advised for small projects or projects with minimal and fixed memory requirements because it is more straightforward and manageable. Dynamic memory allocation, however, is a better choice for bigger projects or for projects whose memory requirements are unpredictable or constantly changing.

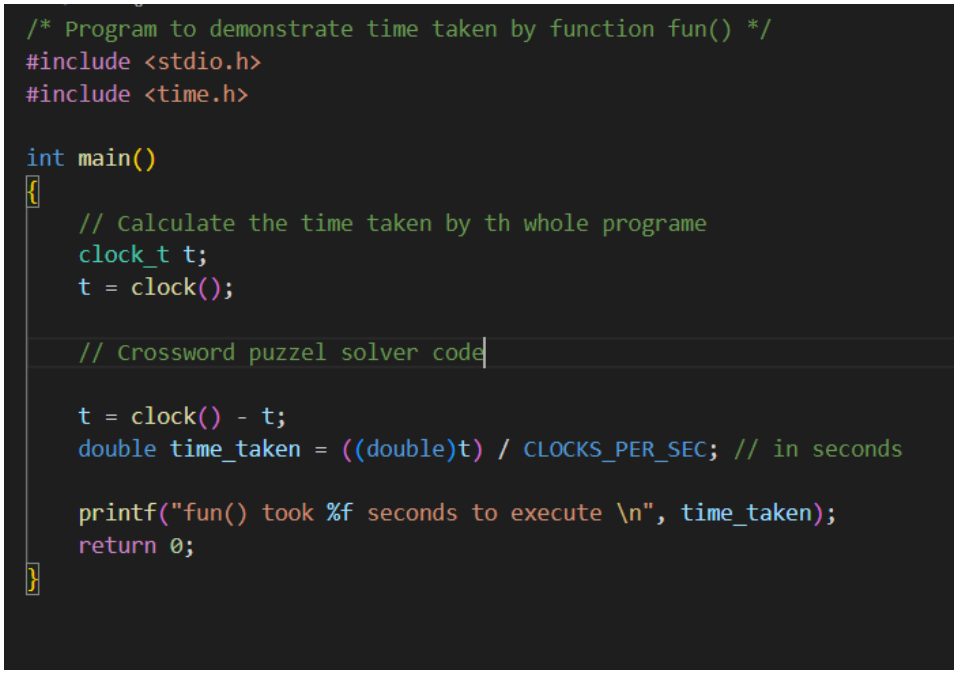
It's also important to note that garbage collectors, which automatically manage memory allocation, deallocation, and memory leaks, are available in current programming languages like C++, Java, and Python, making it simpler for developers to manage memory.

Execution time differences

In terms of execution time, static memory allocation in C is typically quicker than dynamic memory allocation. This is so that there is no overhead associated with allocating and deallocating memory during runtime because static memory allocation is done at the time of compilation.

On the other hand, runtime memory allocation and deallocation for dynamic memory allocation take more time. This is due to the fact that the software must ask the operating system for memory, which might take some time depending on the amount of memory that is requested and the amount of memory that is available. Dynamic memory allocation also leads to memory fragmentation, which can lengthen the time needed to reach a particular memory address.

This is the code snippet, which is used to calculate the execution time



```c
/* Program to demonstrate time taken by function fun() */
#include <stdio.h>
#include <time.h>

int main()
{
    // Calculate the time taken by th whole programe
    clock_t t;
    t = clock();

    // Crossword puzzel solver code

    t = clock() - t;
    double time_taken = ((double)t) / CLOCKS_PER_SEC; // in seconds

    printf("fun() took %f seconds to execute \n", time_taken);
    return 0;
}
```

Execution time for static memory allocation

```
e19300@aiken:~/CO222/MiniProject$ echo '******#***
**########**
******#***
******#***
***#######*
******#*#*
******#*#*
********#*
********#*
********#*

ICELAND
MEXICO
PANAMA
ALMATY'| ./puzzel-static
******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*L*
******D*M*
********A*
********T*
********Y*
Execution time is: 0.000206
e19300@aiken:~/CO222/MiniProject$
```

Execution time for dynamic memory allocation

```
e19300@aiken:~/CO222/MiniProject$ echo '*******#***
**########**
*******#***
*******#***
***########*
*******#*#**
*******#*#**
*********#**
*********#**
*********#**

ICELAND
MEXICO
PANAMA
ALMATY'| ./puzzel-dynamic
******I***
**MEXICO**
******E***
******L***
***PANAMA*
******N*#**
******D*#**
*********#**
*********#**
*********#**
Execution time is: 0.000380
e19300@aiken:~/CO222/MiniProject$
```

The difference in execution time between static and dynamic memory allocation is typically negligible and may not be a serious issue in most situations, especially for short programs. Static memory allocation is appropriate for programs with known and stable memory requirements, whereas dynamic memory allocation is appropriate for applications with uncertain or fluctuating memory requirements. This is the major distinction between static and dynamic memory allocation.

It's also important to note that garbage collectors, which are built into many modern programming languages like C++, Java, and Python, automatically manage memory allocation and deallocation, eliminating the execution time difference between static and dynamic memory allocation. The execution time gap between static and dynamic memory allocation is also lowered by the usage of caching technologies in current systems. Dynamic memory allocation can perform better thanks to caching technologies that store frequently used memory in faster cache memory.

In general, the choice between static and dynamic memory allocation in C should be based on the particular requirements of the program because the execution time difference between the two is typically not appreciable. When selecting whether to utilize static or dynamic memory allocation, factors including the size and complexity of the application, the quantity of memory needed, and the necessity for memory flexibility should be taken into account.