TASK 1

What can C programming language be used for?

The C programming language is also known as the mother of all programming languages since many of the modern programming languages are based on C. According to (testbook, 2022) and (S,R,A, 2023), C programming language is a procedural general-purpose level language that is often used for creating different kinds of applications. This programming language is a very flexible and very popular choice that is frequently used in various applications like embedded systems, operating systems, GPU, and many other things including game development, network programming, system drivers or system programming. C is a very popular choice for these demands because it is effective and has direct interface with hardware because of its simple syntax, and low-level access to memory system and resources. Because of the effectiveness of this programming language, even the C compiler was coded using C language.

Another great thing about this is that Linux OS itself is mainly written in C where only around 2% were written in Assembly language, and according to Munoz (2015), about 97 % of the top 500 most powerful supercomputers run the Linux kernel. C programming is also important for Cybersecurity because knowing how to read and code in C helps a lot when reverse engineering applications using ghidra and other tools. This is because when they need to learn how the application works, they need to dissemble the code and convert it to assembly language. Some people do not know how to read assembly codes, that's why C comes into the picture. A lot of tools convert assembly code to C to help engineers analyze the software better which also comes in handy for gaming developers who want to learn more about the application, (Umrao, 2021).

Who is Dennis Ritchie and what is he known for?

Dennis Ritchie was an American Computer Scientist who a considerable contribution to the field of information technology and computer programming. He was the co-creator of the Unix operating system and is mostly known for his creation of the mother of all programming languages, the C programming language. He did this at the Bell Labs together with his co-creator Ken Thompson in the early 1970's and quickly gain popularity because of its ease of use, effectiveness, and the ability to directly communicate with the computer hardware, especially the memory (Hosch, 2023). The Unix operating system was essential in the early days of computer system development. Many of these ideas that were inspired by them are still in use today and are present in a lot of operating systems.

Dennis Ritchie helped create a number of software and system tools that are still in used by many programmers even up to this day. The C programming language book, also known as "K&R C" is widely regarded as the definitive treatment of the C language and had an impact on several generations of programmers. Additionally, Dennis Ritchie was also an influence in the development of the B programming language, which was the predecessor of C programming language (Hosch 2023).

Explain what sudo command does in Linux

In the Linux terminal, the sudo or super user do command enables the user to run commands with elevated privileges. This is like the administrator or superusers in windows computers. Depending on you settings in the /etc/sudoers file, you can run a single command or commands as a root or as another user. The sudo command was developed as a way to grant a user elevated rights or privileges temporarily. The usage of this command is to simply put “sudo” before the command that asks for superuser permission (Aleksic, 2022).

This is very popular in Cybersecurity where hackers always need this elevated privilege to be able to run admin commands in an OS like Linux. Using this command, a user can complete an operation that calls for elevated rights, like manipulating files, adding a user, changing permissions, changing passwords, deleting files, or even accessing private files. The sudo command is most commonly used to upgrade or update packages in Linux OS since normal users do not have these elevated privileges. The reason is because incorrectly using the sudo command can lead to serious harm to the OS or even web servers, (Sheldon, 2023).

TASK 2 Documentation

Planning:

What I had planned first was to just include all the source files to solve the task easily, but what I learned in this semester is that including source files directly is not a good coding practice, that's why I made header file for each source file.

In this task I added 3 numbers in the original text file to test if the functions were working correctly. The numbers I added were "1", "16", and "6". This is to test if the functions like "isSquared", "isPerfect", and "isCubed" were working. After finding out that the functions were indeed working, I started to fix my program and make it decent.

Coding / Design:

*Coding standards I used: 3 spaces for tab, max 80 margin, used both /**/ and // for comments, Hungarian Notation on my variables as much as possible, and camelCasing for my functions. I also added a simple message using @echo in my makefile to let the user know if the file was successfully compiled. Most of my codes were inspired from this semester and some were also frome websites like stackoverflow or geeksforgeeks.*

At first, I wanted to use the old K&R style of defining Boolean variables, by using struct and defining 0 and 1 to true and false at the header file, but at the end I just used what the tasked has already given me.

To solve this task, I had to create a struct that is based on the task and called it METADATA. After making this struct, in order to print the number in binary I also had to make another function which I called "writeBinaryNumber". Together with the functions from the task file, and my functions, I was able to complete this task and print a nice output.txt which has the right information about if the number has matching mathematical characteristics.

In this task I made sure to not use unsafe functions like gets, memcpy, strcpy, etch. In this task I used fprintf to print to the output file. I also made sure that I initialized my pointers and local variables to 0 or NULL before using them. I also used malloc to allocate memory for the pointers that I used, and later freed them using free() function in C.

Issues:

I wanted to use a debugger in this program but ended up not to, mostly to save time. When I had problems In the code, I just made my own print statements to know where the program failed.

Conclusion:

Overall, I am satisfied with the outcome of my program. It checks the number based on the functions and prints it accordingly based on the struct METADATA. I also got zero errors and zero memory based on the valgrind result when I ran the program with valgrind.

Please see the attachment for the valgrind result.

TASK 3 Documentation

Planning:

What I planned at first was to get inspiration from my previous solved tasks to fully understand how double linked list works. After I got to understand more of the techniques used in making functions, I then implemented these on my program. The plan was to make a flight registry app where the user can register number of flights and add a passenger in it. I decided to call the source file of my double and single linked list function as "database.c" since everything that has to do with creating flight registry, adding passenger, sorting passenger, or deleting flights/passengers is in this source file, while my user input validation functions are stored in my "inputvalidation.c" source file. I got a lot of inspiration from the Cashier Register program, as well as some functions from this semester's homework which were about how pHead, pTail, pPrev, and pNext pointers were used to easily navigate through the list, and how to create, add, or delete nodes from the list.

Coding / Design:

*Coding standards I used: 3 spaces for tab, max 80 margin, used both /**/ and // for comments, Hungarian Notation on my variables as much as possible, and camelCasing for my functions. I also added a simple message using @echo in my makefile to let the user know if the file was successfully compiled. Most of my codes were inspired from this semester and some were also from websites like stackoverflow or geeksforgeeks.*

In order to solve this task, I had to make 3 structs for my lists. Struct FLIGHT and FLIGHTDATABASE was made to handle my double linked list, while struct PASSENGER was for single linked list. I mostly used while loops to iterate through the list, and as well check the input from the user. Since the task is about flight registry, I ended up with my own specific format about the program. In flight ID I chose to have 3 letters followed by 3 numbers just like in booking references ex. SAS153. In destination I chose 3 letters ex. OSL, TKY, or CPH just like at the airports. I also made the age only between 18 and 99, the passenger has to be at least 18 years old before the user can add the passenger in the flight, and I made sure that there are only 1 to 100 sitting spaces at the plane. In this task I based my switch case and main function from the cashier register and made sure to use safe functions and initialize my pointers and local variables to null. It could have been easier to use global variables but did not use them since they are not "safe".

Issues:

This task felt very big and time-consuming. I had to make sure that the program would run perfectly even with unexpected user behavior and make sure that the program doesn't allow stack or buffer overflows. I had a lot of ideas on how I could have made this program better, but since there were more tasks left, I had to move on and just make sure that my program is working fine. The second issue was how to make this program run with unexpected user behavior. I had to make another source file with a lot of user input validation function to make sure that the user cannot enter whatever the user wants.

Conclusion:

Overall, I am very happy and proud of the result since it handles the user input very well, and there are no errors nor memory leaks when running the program on valgrind.

Please see the attachment for the valgrind result.

TASK 4 Documentation

Planning:

The first thing I did before solving this task was to look at my previous solved tasks including the solved threads task from last year. I took some inspiration from this semester's codes and applied them to my simple multi-threaded program. Since the original code from the task is using global variables, I had to think about how I can pass them through my "threads.c" source file. I ended up making a struct which holds these variables and passed them around using pointers.

Coding / Design:

*Coding standards I used: 3 spaces for tab, max 80 margin, used both /**/ and // for comments, Hungarian Notation on my variables as much as possible, and camelCasing for my functions. I also added a simple message using @echo in my makefile to let the user know if the file was successfully compiled. Most of my codes were inspired from this semester and some were also from websites like stackoverflow or geeksforgeeks.*

To solve this task, I first changed the global variables to local variables using my THREADS struct, and then created the main thread that passed the variables into my threads source file. After making sure that my threads were accessible and running, I then started to implement the various functions that reads the given text file in binary and count the printable ASCII characters and specific words like "Hamlet" from the file. To test that the functions were working, I created my own test file and saw that it was matching the words inside the file correctly. I also implemented a run time check using library "time.h" to see how fast the program is runs with my threads running.

In this task I made sure to not use unsafe functions like gets, memcpy, strcpy, etch, and made sure that I initialized my pointers and local variables to 0 or NULL before using them. I also used malloc to allocate memory for the pointers that I used, and later freed them using free() function in C.

Issues:

My biggest issue with this task was that when running my program with the Hamlet text file, my program was not 100% accurate. My conclusion then was that maybe my program was not properly reading the file since it is a huge text file. Maybe my program is splitting the words in the buffer that's why some of the words are not accurately counted. I even double checked with python to see if it also detects the count of the word "Hamlet", and also tried the CTR+F tool in my gedit, Firefox and chrome to see if I get the same result. After updating my countThread function to match case sensitive words using strcasecmp function, my program became more accurate, but still not as accurate as the one from Firefox. My program detects "104" counts of "Hamlet" while Firefox detects "106" whole word / case sensitive Hamlet, but they both detect "95" counts of the word "at".

Conclusion:

Overall, I am proud and satisfied with this program even though it was not as accurate as I thought it would be. The program runs pretty fast with "0.009522 " seconds elapsed time, and as well no errors nor memory leaks from the valgrind result

Please see the attachment for the valgrind result.

TASK 5 Documentation

Planning:

The first thing I did when starting on this task was to check how the netcat tool works. Since the task is about reverse shell, I made plans and sketches on how my server-client program would be like. Since the task involves having the server and client codes in the same program, I ended up making a struct which passes my iClientSocket and iServerSocket variables, instead of using global variables. After figuring out how to run both server and client from the same program just like in netcat, I had to figure out how to send a signal to both if either has terminated the program.

Coding / Design:

*Coding standards I used: 3 spaces for tab, max 80 margin, used both /**/ and // for comments, Hungarian Notation on my variables as much as possible, and camelCasing for my functions. I also added a simple message using @echo in my makefile to let the user know if the file was successfully compiled. Most of my codes were inspired from this semester and some were also from websites like stackoverflow or geeksforgeeks.*

To solve this task, I had to change the given code and change some functions to make my reverse shell application more authentic. In my execute command code, I used while loop to iterate over the client output and append it to my buffer. In this way, if the user types "ls" it will print all the files in the given directory. I also added extra feature that tells the user if changing the directory was successful or not. This was not possible earlier using the given reverse shell code. In terms of design, I made the program look as authentic as possible as if the user is using netcat.

To handle my signal function in my program, I used sigaction function instead of the older signal function. This was because Sigaction is considered safer and provides more fine-grained control over signal handling than the older version. In this task I made sure to not use unsafe functions like gets, memcpy, strcpy, etch, and made sure that I initialized my pointers and local variables to 0 or NULL before using them. I also used malloc to allocate memory for the pointers that I used, and later freed them using free() function in C.

Issues:

I almost made it as if the client was the one sending the commands, and not the server. I had to read the task multiple times and then understood at the end that the server will send the commands over to the client, and then the client will execute these commands and send the output back to the server. One issue I also had was that the original execute command function only prints 1 line of output. I had to remake this function so that it works as if the user is sitting in front of the terminal. The problem I had at the end was that if the server terminates the program, the client will automatically terminate, but if the client terminates the program, the server needs to input something before it terminates because my server is using while loop and is waiting for an input.

Conclusion:

Overall, I am very satisfied and proud of this task since I was able to make the program work like a simple version of netcat. The user can change directory, make, or write to a file, or even do cat /etc/passwd. When running the program in valgrind, it returns 0 errors and memory leaks as well.

Please see the attachment for the valgrind result.

TASK 6 Documentation

Planning:

Before I started coding for this task, I first had to do a lot of research if there are codes that does the converting functions that might be helpful. I found out that regex function in C can search for regular expressions or search for a pattern in text, but later found it can be unsafe that's why I dropped the idea. What I did then was just test some other codes and functions that might work and ended up making pointers for while loop characters and unsigned int variables that I can search for using strchr() and strstr() function in C.

Coding / Design:

*Coding standards I used: 3 spaces for tab, max 80 margin, used both /**/ and // for comments, Hungarian Notation on my variables as much as possible, and camelCasing for my functions. I also added a simple message using @echo in my makefile to let the user know if the file was successfully compiled. Most of my codes were inspired from this semester and some were also from websites like stackoverflow or geeksforgeeks.*

To solve this task, I had to make and test the functions 1 by 1 and make sure that they were working before I added the rest. I included the simple source file that I made to test if my program was applying the functions correctly. What I did first was to apply the "replaceThreeSpacesWithTab" function and see if it works. After making sure that the program is replace three spaces with a single tab, I then added the "applyHungarianNotation" to see if it renames my variables. My program was able to change 3 spaces with a tab and apply Hungarian notation to unsigned int variables by using a temporary pointer and strstr() function in C. It is important to remember that the source file I had to make will not even be compiled, I just made to test if my program can detect unsigned int and 3 spaces even if inside a while loop. After making sure that the two functions work correctly, I finally added the last function which converts while loops to for loops and saw that my program was able to convert my test source file successfully.

In this task I made sure to not use unsafe functions like gets, memcpy, strcpy, etch, and made sure that I initialized my pointers and local variables to 0 or NULL before using them. I also used malloc to allocate memory for the pointers that I used, and later freed them using free() function in C.

Issues:

The biggest Issue I had was that my while loop to for loop function is not compatible with complex while loops. Sometimes it bugs and sometimes it does not work at all. If the source file is simple, all three functions will work correctly, but if the while loop is complex, only the "replaceThreeSpacesWithTab" and "applyHungarianNotation" function will work. I also tried to include the ">" greater than sign for my program to search for, but I got some bugs where it suddenly produced duplicate codes and made the output file worse than before.

Conclusion:

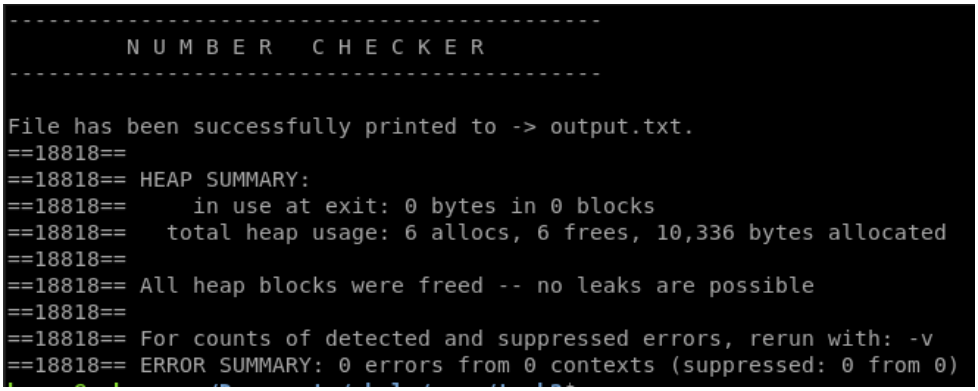
Overall, I am satisfied to be able to answer this task, but because of these issues, I just made my program as simple as possible. The program has 0 errors and memory leaks according to valgrind.

Please see the attachment for the valgrind result.

ATTACHMENTS

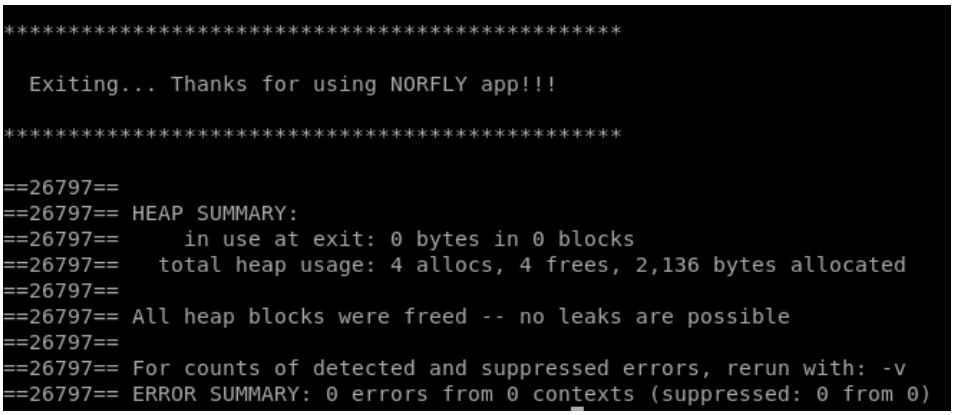
Task 2

Valgrind result: 0 errors and memory leaks



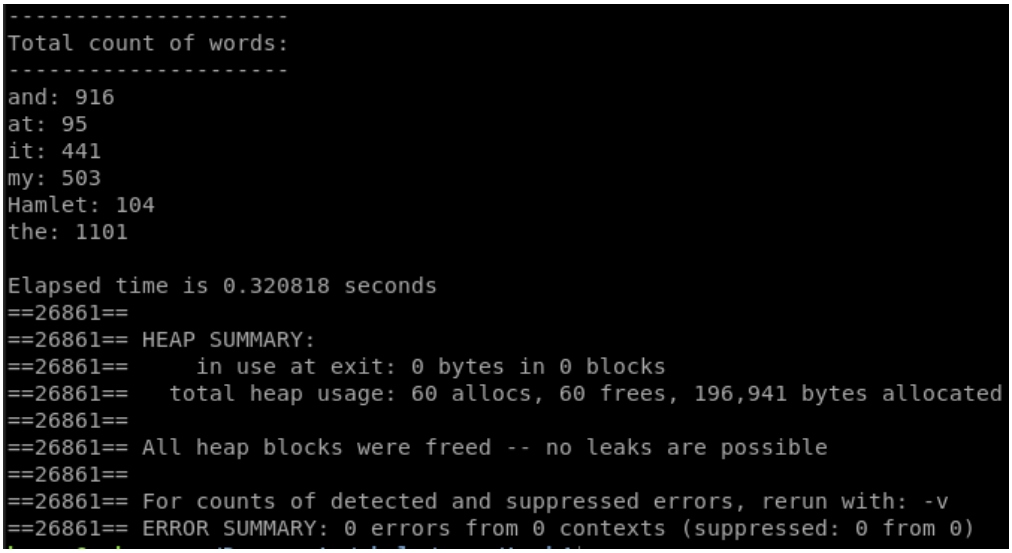
Task 3

Valgrind result: 0 errors and memory leaks



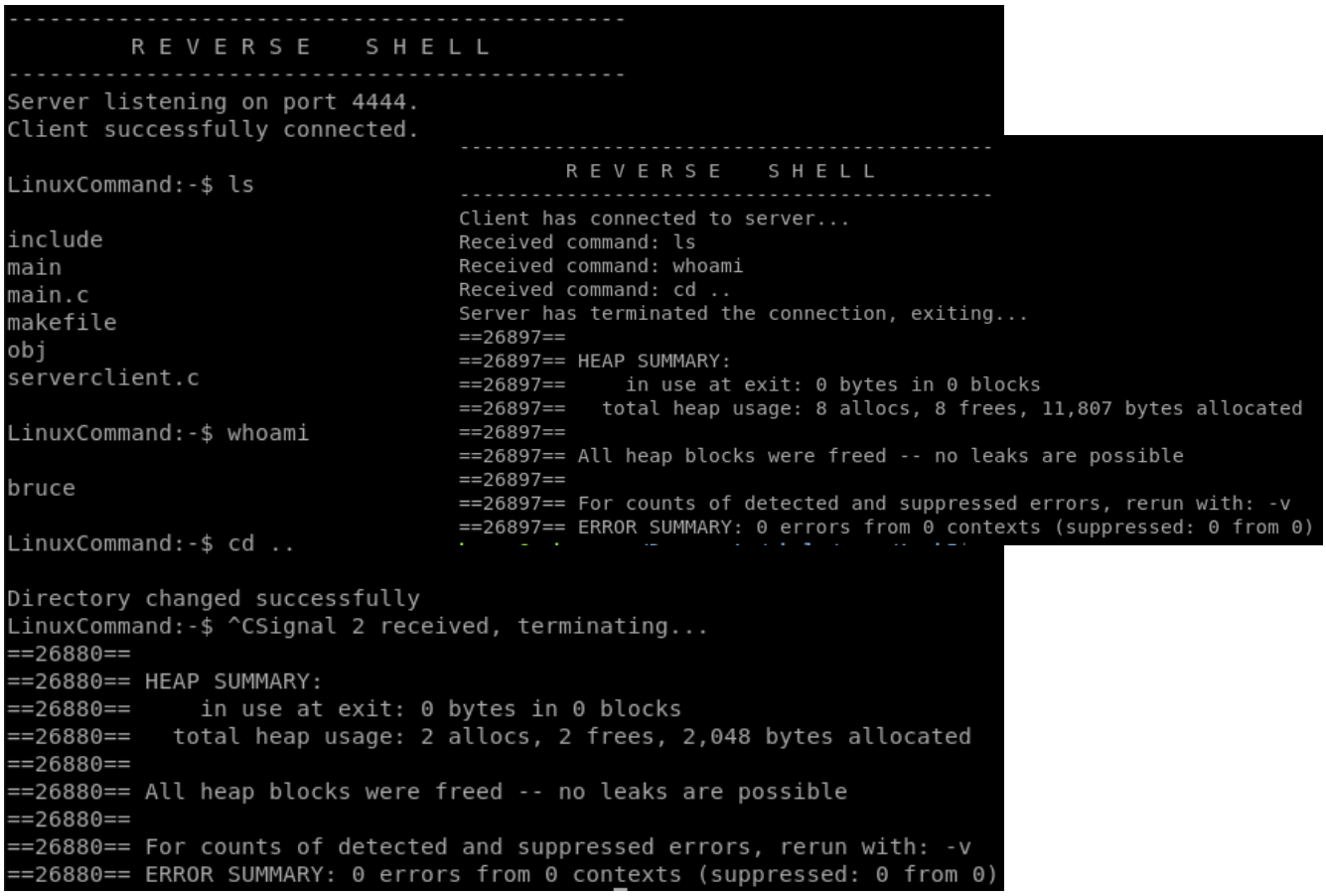
Task 4

Valgrind result: 0 errors and memory leaks



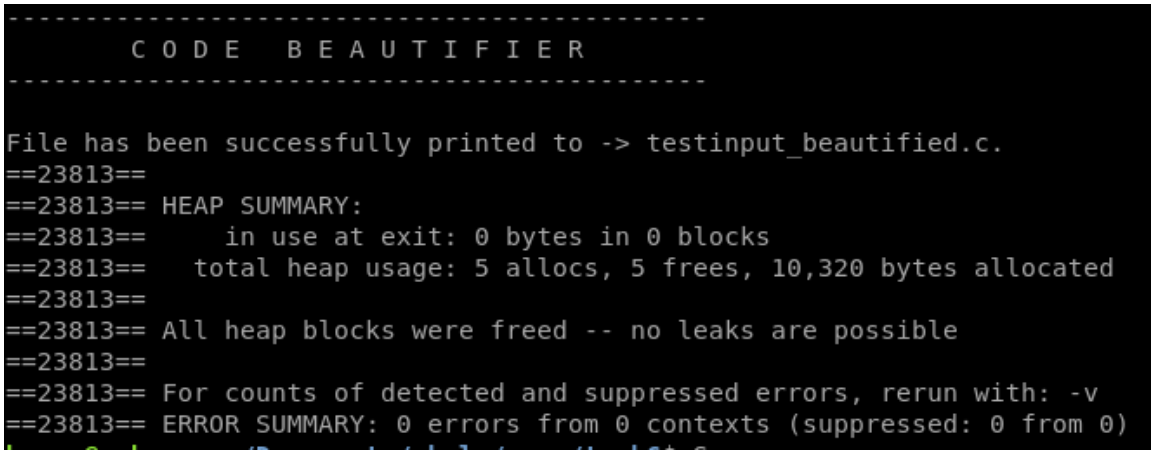
Task 5

Valgrind result: 0 errors and memory leaks



Task 6

Valgrind result: 0 errors and memory leaks



SOURCES

Munoz, D. (2015). After All These Years, the World is Still Powered by C Programming. Toptal Engineering Blog.

https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

Umrao, R. (2021, December 14). Reverese Engineering: Understanding a C program - Rishabh Umrao - Medium. Medium.

https://ayedaemon.medium.com/reverese-engineering-understanding-a-c-program-b4bf5ec7def6

Testbook Which programming language is called the mother of programmi. (2023, April 10). Testbook.

https://testbook.com/question-answer/which-programming-language-is-called-the-mother-of--5efebba3f835040d12d83215

S, R. A. (2023). Use of C Language: Everything You Need to Know. Simplilearn.com.

https://www.simplilearn.com/tutorials/c-tutorial/use-of-c-language

Wikipedia contributors. (2023). Dennis Ritchie. Wikipedia.

https://en.wikipedia.org/wiki/Dennis_Ritchie

Hosch, W. L. (2023, April 14). Dennis M. Ritchie | Biography & Facts. Encyclopedia Britannica.

https://www.britannica.com/biography/Dennis-M-Ritchie

Sheldon, R. (2023). sudo (su 'do'). Security.

https://www.techtarget.com/searchsecurity/definition/sudo-superuser-do

Aleksic, M. (2022). How to Use the sudo Command in Linux. Knowledge Base by phoenixNAP.

https://phoenixnap.com/kb/linux-sudo-command