Assignement 2
C + Python Programming

gregor.jouet@devinci.fr

1 Requirements

1. You must have a working Ubuntu (or other Linux based) system installed **PRIOR** to this class. Failure to do so will prevent you from doing this assignment.

2. It is recommended, although not required, to work on this assignment in advance and to use the actual assignment day to ask questions and engage in discussion on the topic.

Contents

2 Foreword

Although regarded as the primary outcome and skill of a computer scientist, programming is a tool and not a end in it of itself. It is however omnipresent and encountered daily by many professionals in the industry and lays the foundations of the ways automated systems operate. Programming languages are designed for humans and are meant as a tool to bridge the gap between the machine and human comprehension. Like all the other lessons in this class, the goal is to understand the reasons for a particular design choice and to understand what action is performed for each instruction given to the processor.
The elements a programmer interact with the most are the languages themselves, but the syntax or singularities of a language must not distract you from the fact that they are only the visible part, the front end, and that each language carries a lot of depth past this front.
Your goals are to familiarize with as many languages as possible and to do as many exercices as you can in each section.

3 C Programming

The next section is splited by difficulty level, you will start by setting up your environment and will gradualy tackle more difficult problems.[1]

3.1 Pre-Flight

1. Write a C program that write 'Hello World!' in the terminal.
2. Use a Makefile [2] to compile and clean your project folder.
3. Use a Makefile and a CMakeLists.txt to manage yoru project. The Makefile must have a clean, build and run target.
4. Make a program that accepts a string on stdin and rewrites it on stdout.
5. Make a program that reads a string on stdin and outputs it on stdout only if an integer was inputted. (Use the **atoi** function)

3.2 Warmup

1. Write a C program to calculate the distance between the two points.
2. Write a C program to read an amount (integer value) and break the amount into smallest possible number of bank notes.
3. Write a C program to convert a given integer (in seconds) to hours, minutes and seconds.
4. Write a program that reads two numbers and divide the first number by second number. If the division not possible print 'Division not possible'
5. Write a C program that reads two integers p and q, print p number of lines in a sequence of 1 to q in a line.
6. Write a C program that reads an integer and find all its divisor
7. Write a C program that reads the side (side sizes between 1 and 10) of a square and prints square using hash (#) character.
8. Write a C program that reads the side (side sizes between 1 and 10) of a square and prints a hollow square using hash (#) character.
9. Write a C program to display the sizes and ranges for each of C's data types. (int, float, uint, double, char)
10. Write a C program that accepts an integer and print next ten consecutive odd and even numbers.
11. Write a C program to calculate the sum of two given integers and count the number of digits of the sum value.

[1] A portion of this section comes from the W3C [1] programming exercices references. Feel free to do the rest of the exercies. The ones here have been cherry-picked.

12. Write a C program that generates 50 random numbers between -0.5 and 0.5 and writes them in a file rand.dat. The first line of ran.dat contains the number of data and the next 50 lines contains the 50 random numbers.
13. Write a C program that accepts an integer n and outputs the factorial of n using recursion.

3.3 Intermediate

1. Write a program that outputs the size of an int variable, the size of an int* variable.
2. Using the & operator, write a program that outputs the address of an int variable in the stack, and the address of one in the heap. (Use %p in printf) compare it to a map layout of a process in /proc/[pid]/maps
3. Using the **sbrk(2)** syscall find and print where the heap actually ends.
4. Write a program that allocates an integer in the heap using malloc
5. Write a program that allocates 40 integers in the heap and sets them all to 0.
6. Use the **memset** for the above question if you have not.
7. Write a C program that takes a filename as output and print its size in bytes in the terminal.
8. Write a C program that takes a filename as output and print its contents in the terminal.
9. Double Pointers: Write a program that reads user input, and returns the sum of the string lengths that were inputted. The program computes the length when an empty string is supplied. To store the strings you may have to use double pointers (pointers to pointer).
10. Create a function with the signature 'void my_func(int a)' that displays the number a. Then create a pointer to this function, call this function by dereferencing the pointer you created (use a static argument)
11. Declare a Point structure that can hold 2 float.
12. Initialize 2 of Point structures in the stack. Then in the heap.
13. Write a program that asks the user how many points he is going to enter, then read each x and y of these points, then compute and output the centroid of this set.
14. Write a program to manage a library: From a main menu you can choose to list the books, add a new one made of an id (int) and a title or to delete one. You cant have two books with the same id
15. Use the **open(2)** syscall to create and open a file. Use **write(2)** to write Hello inside of it. Observe the syscalls with strace.

3.4 Forks, Threads and process flow control

1. Setup your code and compilation options to use the pthread library.
2. Use **pthread_create** to create a new thread that will wait 5 seconds before returning.
3. Use **pthread_create** to create two threads that increment the same counter until it reaches 100. Print the counter each step. What happens ?
4. Use **pthread_mutex_lock** to create a critical section where only one thread has access and increments the counter. Dont forget to unlock
5. Create a program that initializes an array of 100 integers from 1 to 100 and distributes the tasks of calculating the inverse of each of these numbers in a new memory area (of doubles). Make the numbers of integers and the numbers of threads adjustable and print out the time each thread spent calculating as well as the total time of the program. Try different values.
6. Fork a process using **fork(2)** and print the child process and parent process pid using the fork return and **getpid**
7. Write a program that forks to match the diagram in Figure 1, where each box is a process and each arrow is a fork.
8. Write a program that exits after 3 Ctrl + C instead of 1. Use **sigaction(2)**

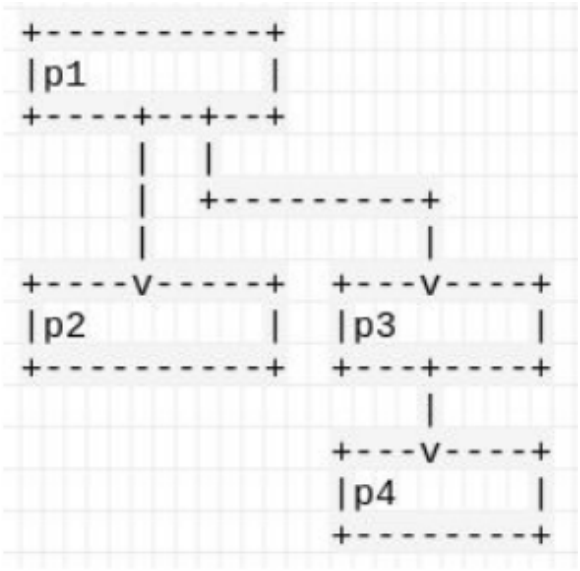

Figure 1: Fork processes in this manner

3.5 Problems

1. Write a program to convert a base 10 number to any base between 2 and 16, the base should be provided as an argument.
2. Write a program that outputs the total size of files in a folder. The program should walk the fodlers recursively. As a bonus, make this program output the size in a human readable form.
3. Write a program to multiply matrices. The user should be asked the dimensions of the matrices, to input each element and the matrice be printed.

3.6 To Go Further

These problems may take you from a couple of hours to a week each, depending on your familiarity with all the concepts seen above and your proficiency level in C.

1. Use the **libfuse**[2] library to mount a FUSE filesystem that exposes a single file named 'update' that executes **sudo apt update** when you read from it.
2. With the **sbrk** syscall, explain, then implement how to make your own **malloc(3)** and **free(3)** functions. How would you do it with the mmap syscall?
3. Write your own **printf(3)** function using **write(2)** and Variable Arguments. Handle %i and %s at first. Use itoa or sprintf if it is not available. Do the same for the **sprintf** function.

[2]Use an example file from https://github.com/libfuse/libfuse as a starting point.

4 Python Programming

4.1 Pre-Flight

1. Check your python version.
2. Install the numpy package using pip.
3. List the currently installed packages
4. What is a requirements.txt file ?

4.2 Write Python

1. Make a python script that outputs Hello World
2. Make a python script that prints integers from 0 to 100
3. Make a python script that reads a value from the user using input and prints it
4. Make a python script that reads a value from the user and outputs Hello Python as many times as the user entered.
5. Make a python script that reads a value from the user and outputs its log value, if the number given is i= 0, the script outputs nothing.
6. Create a python project (just. . . a new folder, an empty requirements.txt and a README)
7. Create a test module, which is a folder and a test_module.py in it.
8. In the root of your project create a main.py file which import your test_module
9. Import the math module, the numpy module, make an alias for this import as np
10. What happens if you put a hello world in the __init__.py file ?
11. Write a program that reads numbers from the console until an empty line is entered, then outputs the numbers backward.
12. Write a program that reads numbers from the console until an empty line is entered and outputs the number of times each number has been entered.
13. Write a program that reads numbers from the console until an empty line is entered and outputs the numbers from the lowest to the highest value.
14. Write a program that reads a sentence and outputs the number of occurrences of each character.
15. Write a program that checks of two lists have at least 1 element in common.
16. Write a program that converts a list of single characters to a string
17. Write a program to insert a string at the beginning of every item in a list
18. Write a program to merge two dictionaries.
19. Write a program that removes even numbers from a list of integers
20. Write a python function that returns the square value of the passed argument if it is ¿0 or the exponential of the argument otherwise
21. Modify the function to return both values in a tuple and unpack the returned value in the calling script.
22. Write a function that takes an arbitrary amount of arguments and returns the sum of these arguments.
23. Modify the function to return the sum, mean and standard deviation of the passed argument list.
24. Write a function that returns the min and max value of an arbitrary list of arguments.
25. Write a program that creates a file and writes what the user inputs in the console to it, until an empty string is entered.
26. Write a program that reads a file where each line is a number and outputs the sum of the read numbers

27. Using **json.dumps** from the json module, write a program that reads a string from the console and writes to a file the number of occurence of each character in a JSON format.
28. Write a program that creates a folder, reads a number n from the console, then creates files 1 to n in the new folder, each file containing the This was created by python string.
29. Write a program that reads the /proc/self/maps file and outputs a list of information about each mapping, including the memory range, protection and name.
30. Using os.stat and os.isdir or os.isfile and a recursive function, write a program that scans a given directory and outputs the path of the 5 biggest files, and the total size of the scanned directory.
31. Write a function to get an element from a dictionary, if the Key is not in the dictionary, make the function return None. Write this function with and without a **try except** block.

4.3 Introduction to Object Oriented Programming

You are tasked with the realisation of a Hotel Management System. Your program has to keep track of customers, rooms and their occupation planning. The program should be the one used by the reception clerk and have the abilities to register a booking and show the booking calendar for each room.

1. Draft the different classes required for this system and their interactions.
2. Write your classes in Python
3. Add a menu to add a new room, a new customer and when they will be occupying the room.
4. Make the program read and write the current state of the hotel to a file, likewise, add a function to load the state from a file.

4.4 Writing pythonic code (DONT SKIP THIS PART)

4.4.1 Dunders

1. Using the __**getitem**__ and __**setitem**__ functions, make a class similar to a dictionary but that returns None instead of throwing an error when the key is not found.
2. Using the __getitem__ function, write a class that returns the file contents of the given argument: a[file.txt] returns the contents of file.txt or None if file.txt does not exist.
3. Add a __str__ function to a class and pass the class in print directly.
4. Add a __del__ method that prints Goodbye and delete all references to the class.
5. Create a Matrix class, using __getitem__ __setitem__ implement a way to access or set matrix elements. Using __add__ __mul__ implement matrix multiplication and addition, throw an error if the matrices size do not match.
6. What are the __enter__ and __exit__ functions used for ?

4.4.2 Generators

1. Write a generator that returns each character of a given string.
2. Write a Fibonnaci number generator.
3. Write a generator that returns each file in a given folder

4.4.3 Ternary Operator

1. Using the ternary operator, make the absolute value function.
2. Using the ternary operator, make a function that return the log value of the argument or return None if the argument is i= 0
3. Put the two function above im lambda epressions

4.4.4 List comphrension

1. Write a list of squared numbers from 1 to 100
2. Given a string, make a list of the length of each word in the string.
3. Given a list of floats, make a list of integers from this list which contains only the positive numbers.
4. Do the previous question with a **filter** and a **lambda** expression.

4.5 Problems

4.5.1 Sound manipulation

1. Use the **numpy.linspace** function to create a numpy array of 100 numbers from 0 to 100 spaced by the same increment
2. Using the Matplotlib **plot** and **show** function, show the graphs of the square function, the inverse and log functions.
3. Using the **multiprocessing.Pool** class, distribute the calculation of the nth prime number. You can use a function to determine if a given number is prime or not.
4. Using the **imageio** module, read an example image and show it with matplotlib
5. Print the shape of the numpy array representing the image, nullify the red channel and print the image again.
6. Use numpy to generate a 440Hz sine wave, use the **simpleaudio** module to play this note. You will have to convert the sine wave to **int16** by first, scaling your [-1.0, 1.0] sine wave to [-32 768, 32 767] and then convert your data using np.astype. Then use **simpleaudio.play_buffer**. (Decide a sample rate, which is the number of points per second, before creating your sine wave a sample rate of 44100, 44kHz works well)
7. Likewise you can use **sounddevice** module to record your microphone and use matplotlib to show what your voice sounds like.

4.5.2 Keys Per Second

1. Install the **pygame** package
2. Open a window using pygame.
3. Get a text file with different lines in plain english that will be used for writing.
4. Make the first line display on the screen
5. Read the user keyboard input from the window and show it in the terminal, then on the screen
6. Show when the correct or incorrect key is pressed and then measure the keys per seconds and display it on the screen.

References

[1] *C Programming Exercises, Practice, Solution*. URL: https://www.w3resource.com/c-programming-exercises.

[2] *Makefile tutorials*. URL: https://makefiletutorial.com/.