—Assignment #1—

If you are reading this either the day before or the same day this assignment is due:

- the only way to earn marks now is if you write test code for the others members in your group
- no marks will be earned for a copy-paste or refactor of someone else's code
- test code submitted will have a 10% penalty applied for not fully participating in your group

The basic work to do for test code:

- check each of the functions of the others group members code
- if the others have all code in the same function, then split into reasonable smaller functions yourself
- test the organized functions with enough variety of input to see that they work correctly
- use a text file to log the results of executing your tests with time and date for each test session

If there is not code submitted from others in your group, then that is your choice to gamble working on assignment last-minute.

Remember, you must have a git log of your progress to earn full marks. Commit to your git at least every other day, and break down your work into small pieces. Check Blackboard group for messages.

Learning Goals

1. practice programming C
2. practice arrays, `struct`, and dynamic memory allocation
3. the above skills support the understanding of various operating systems, in this case, the data similar to the design of data in kernel code that relies on pointers and structures
4. lastly, to practice for when we make calls to the operating system's provided functions

Work the best you can as a group for this first assignment, but submission is individual:

- groups are randomly assigned on Blackboard (check the Groups section);
- collaborate using UFV GitLab as much as you can in your group;
- reference any website you use to help you write *small* parts of your code
  (e.g.: the code used from an outside source should not solve the entire assignment),
- if so, give brief comments describing how you modified something you used and why;
- do not copy an entire program.
- backup your code separately—code only in the group shared repo can easily be lost by others in the group experimenting with git commands; one git clone folder and a different folder of your own.
- make your own separate copy, IN A DIFFERENT FOLDER, and not a subfolder of the clone.

You will be assigned to a group of four. Together, choose a group leader and have them:

- create a project named `MyArgsProject` on GitLab
- add the others to this project (their UFV GitLab username should be on Blackboard discussion)
- set "Developer" role for each person in the project in UFV GitLab (otherwise they cannot git push)
- post your UFV GitLab username in the Blackboard discussion if you have not done so already (make sure to repost if you change it again)

C-string Arrays

Take the code we have practiced with graphs and write a program `checkGraph` that:

- takes arguments typed on the command line together with its execution command
  e.g.: `./checkGraph sorted size`
- copies the arguments to one array of C-strings
  - each element of the C-string array should be one command-line argument
  - dynamically allocates the array of C-strings so that any number of arguments can be passed in to your program
- each command-line argument as an array element itself should be a dynamically allocated array of chars (a "C-string")
- consider each argument a word, with sequence of characters ending in a null character `'\0'`
  (read C library documentation in `<string.h>` header library carefully)
- NO USE of `strtok` function, since it defeats the purpose of the practice you need with pointers

Make a `struct` for each argument, where the type is as follows:

```
typedef struct
{
  char *letters;
} word;
```

Make an array:

- where each element is of this type `word`, and
- initialize the elements to the arguments stored in the C-string array;
- write small functions, say, to manage allocation of memory,
  similar to constructors in OO-programming.
- Remember to free your dynamically allocated memory.

To help yourself check your work, print the array of words to standard output all on one line with a single space separating each pair of words.

Check that your program can deal with a variety of command-line arguments and not just the ones expected for managing execution of the other smaller programs described next.

Once you have command-line copies of C-string agruments in an array, use this to decide whether to execute the following *other* programs:

- FIRST PROGRAM: (if argument C-string `sorted` was written) output a statement whether a graph has its node connections (lists) sorted from smallest label to largest label for all its lists
- SECOND PROGRAM: (if argument C-string `size` was written) output a statement for total connections in a graph (total length of lists and divide by 2, known as the Handshake Theorem)

For both of the above programs, use the graph in your programs as shown in the video demonstrating fully implemented breadth-first-search.

Each of the above programs should be managed and executed by your `checkGraph` program through calls to `fork` and `exec`. We will have discussed a few variations of `exec` in class with at least one small demo to get you started.

It is also possible to execute a bash command in your program if you run it in a Bash terminal with `system("command");` and you can read more in documentation:
https://en.cppreference.com/w/c/program/system

If one of the arguments that was typed into your program matches with the command `"ls"`, then have your `graphCheck` program execute that command.

Debugging

If you can manage to set it up on your own environment, GDB is the GNU Project Debugger. We do not have this installed on Jupyter Notebooks. On your own, if possible, install and then run your program with GDB by compiling your program with the `gcc` option `-g`. GDB can tell you what line segmentation faults (bad memory accesses) happen inside your program.

Various commands for GDB:

- `gdb` begin an interactive session of GDB
- `quit` exit the interactive GDB terminal prompt
- `file <your_program>` load your program
- `run` let GDB execute your program

Submission

Submit your individual git repository as a bundle.

Create a bundle file on the command line using git:

```
git bundle create GraphCheckProject.bundle master
```

inside the `GraphCheckProject` folder. Do not worry about adding this file to your project itself. It is meant to compress the project into one file with the records of your commits. You can check what is in your bundle file with the following command:

```
git bundle verify GraphCheckProject.bundle
```

Submit the bundle file only to our Blackboard Assignment and Tests section in Assignment 1. You can submit multiple times, but I only mark the last submission before deadline. Submit every couple of days, and then you are always worry free to know your submission has most of your hard work.

Please do not email me to check that your files are submitted to Blackboard. You are capable of checking that yourself.

This assignment is due on Thursday, Sep 29, 11:59 pm.

Marking Rubric

20 marks total

	excellent	**good**	**adequate**	**poor**
code style (4 marks)	no mixed whitespace (tabs and spaces), indentation aligns per block of code, meaningful variable names, and no unnecessary lines of code	one or two issues with mixed whitespace, alignment, variable names, or extra unnecessary code	more than two issues with style	many issues with style
code functionality (16 marks)	project has regular (every other day) git log, no runtime errors, word elements stored in a dynamically allocated array, the `ls` command executes when matching one of the arguments, and other small graph programs execute through `fork` and `exec` calls	project has git log with multiple commits spread out over time, no runtime errors, word elements stored in array, and other small graph programs execute through `fork` and `exec` calls	git log with few commits, or a few runtime errors, or variable names are trivial `"a", "b", "c",...`, or output is difficult to read, or memory is not managed properly	no git log, there are runtime errors that halt the program, or output is not presented in a readable format, or the program does not compile