PROGRAMMING IN THE UNIX ENVIRONMENT, DV1457/DV1578

LAB 2: C PROGRAMMING

Modified by Amir Yavariabdi

Prepared by Sai Prashanth Josyula

Blekinge Institute of Technology

September 1, 2024

A. Objective:

The objective of this laboratory assignment is to practice the following concepts in the context of C programming: dynamic memory allocation, pointer manipulation, arrays, file I/O, and implementing the K-means clustering algorithm. You will also gain experience with modular code design, including the use of object files, linkers, and Makefiles to manage and compile a multi-file C project. Specifically, you will apply these concepts to cluster 2-dimensional data points based on user-specified cluster counts and produce an output file with the clustering results.

B. General Requirement:

The laboratory assignments should be conducted, solved, implemented, and submitted in groups of two students.

C. Preparations:

Read through these laboratory instructions carefully and make sure that you understand what you are supposed to do. If something is unclear, please contact the corresponding teacher via aya@bth.se. You are expected to have acquired the following knowledge before the lab sessions:

- You are supposed to have basic knowledge about working in a Unix environment.
- You are supposed to have good knowledge and experience in C programming.
- You should be familiar with the operating systems concepts dealt with in this assignment.

The main topics relevant to this assignment are dealt with in lectures 5–8 of the course and are as follows: *(i) dynamic memory allocation, (ii) pointer manipulation, (iii) array, (iv) file I/O,* and *(v) program management.* You are expected to actively find and learn concepts required to complete this assignment. You do not need to wait for the corresponding lecture to be completed before you incorporate those concepts in your code; you can go ahead, learn the concepts, and start using them already. Do not forget to consult the relevant sections.

D. Collaboration and Plagiarism:

You are expected to work in groups of two. Groups larger than two are not accepted. You are also not expected to work alone unless you have a good reason or permission.

You are allowed to discuss problems and solutions with other groups. However, the code that you write must be your own and cannot be copied or downloaded from somewhere else, e.g., fellow students, the internet, LLM, etc. You may get ideas from various sources, but you must write the code yourself and be able to clearly describe each part or line of your implementation. The submitted solution should be developed by the group members only.

E. Problem Definition, Your Task, and Technical Requirements:

The problem is to implement and apply the K-means clustering algorithm to a set of two-dimensional data points, which are provided in a text file named **kmeans-data.txt**. This task involves multiple steps:

- Reading Data: Extract two-dimensional data points from the text file **kmeans-data.txt**.
- Prompt the user number of clusters and/or allowing user input for initial centroid selection.
- Clustering: Apply the K-means algorithm to partition the data points into clusters. The K-means algorithm is an unsupervised machine learning method that clusters data by iteratively assigning each point to the nearest centroid and updating the centroids based on these assignments. Note that the centroids represent the center points of the clusters and can be initially assigned to random data points or based on user selection.
- Writing Results: Output the clustered data to a new text file, **kmeans-output.txt**. The resulting file will contain the original data points with an additional column indicating the cluster assignment for each point.

To achieve this, you are required to implement two C source files:

- main.c: This file will handle reading input, prompt the user number of clusters and/or prompt the user input for initial centroid selection, apply the K-means clustering algorithm, and writing the results to an output file.
- kmeans.c: This file will contain the core implementation of the K-means clustering algorithm and can include functions for matrix allocation, distance calculation, centroid initialization, clustering, result writing, etc.

Additionally, you will **need to** create a **Makefile** to manage the compilation and linking of these C source files into a single executable binary named **kmeans**.

Steps to Follow

1. **Implement the Source Files**:
    - Write main.c and kmeans.c as described above.
2. **Create a Makefile**:
    - Develop a Makefile to, at a minimum, compile the source files into object files and link them into an executable. The Makefile can be tailored to include additional features such as automatic cleaning of object files, detailed compilation flags for debugging, etc.
3. **Compile and Execute**:
    - Open a terminal in the directory containing the source files and Makefile.
    - Run make to compile the program and generate the executable kmeans.
    - Execute the compiled program by running ./kmeans in the terminal.
    - The program will **read** the data from **kmeans-data.txt**, apply the K-means algorithm, and produce an **output** file, **kmeans-output.txt** with the original data and cluster assignments.
    - The program check for memory leak and debugging.

Note: Memory leak detection can be integrated into the **Makefile** using *Valgrind* or performed manually from the terminal. For debugging, you can use *gdb*. Adding dedicated targets in the Makefile for these tasks simplifies running checks without typing commands each time. **In this assignment, to achieve higher grades**, memory leak detection and debugging must be incorporated into the Makefile (please, check Grading Strategy section).

By following these steps, you will implement and run a K-means clustering algorithm, manage source files and object files using a Makefile, and generate an output file with the clustering results.

F. Main Expectations After Completing Your Code:

Once you have finished implementing and running your code, we mainly expect the following:

1. Correct Implementation: Your program should correctly read the data from **kmeans-data.txt**, apply the K-means clustering algorithm, and output the results to **kmeans-output.txt** with the proper cluster assignments.
2. Code Quality: The code should be well-structured and modular, following good programming practices. Functions should be clearly defined, and memory management should be handled correctly.
3. Makefile: The Makefile should correctly compile the source files into object files and link them to produce the executable kmeans and can be tailored to include additional features.
4. Documentation: Your code should include comments explaining the functionality of key sections, and the Makefile should be documented to describe its purpose and usage.
5. Execution and Output: Running ./kmeans in the project directory terminal should generate **kmeans-output.txt** with data points and their respective cluster assignments. Ensure the output file is correctly formatted and matches the expected results.
6. Error Handling: Your program should handle errors gracefully, such as issues with file opening, memory allocation, or invalid input, and provide informative error messages to assist with debugging.
7. Memory Leakage and Debugging Checks: For those targeting Grade A, ensure to perform memory leak detection using tools like Valgrind and include debugging checks like gdb. These checks must be integrated into the Makefile. Including these steps ensures the program runs efficiently without memory issues and helps identify potential bugs.
8. Environment: Programs must be written in C, compiled using gcc, and run in Unix environments, specifically Ubuntu 24.04 or Fedora 40. Ensure compatibility with these environments during development and testing.

By meeting these expectations, you will demonstrate your ability to implement and manage a clustering algorithm, handle multiple source files and a Makefile, and produce reliable and accurate results. Check Grading Strategy section for more details.

G. Project Structure:

The folder structure for your project must look as follows:

```
project_directory/
├── kmeans              # Executable binary file
├── kmeans.c            # Source file containing K-means algorithm implementation
├── kmeans-data.txt     # Input data file
├── kmeans.o            # Object file for kmeans.c
├── kmeans-output.txt   # Output file with clustering results
├── main.c              # Source file containing main function
├── main.o              # Object file for main.c
└── Makefile            # Makefile for building the project
```

To visualize the project structure in your terminal, follow these steps:

1. Install the tree utility by running the command: **sudo apt-get install tree**
2. Once the installation is complete, use the **tree** command in your terminal to display the directory structure.

This will provide a visual representation of the project's directory hierarchy.

Note: If you do not fully satisfy the specified requirements and expectations or fail to adhere to the project structure, your submission will receive an F grade. It is crucial that your program compiles, runs without errors, and generates the required output for evaluation. In such cases, the teacher will not evaluate your code or provide additional feedback. To avoid losing a submission opportunity, ensure you follow all requirements, expectations, and project structure guidelines.

H. K-means Clustering:

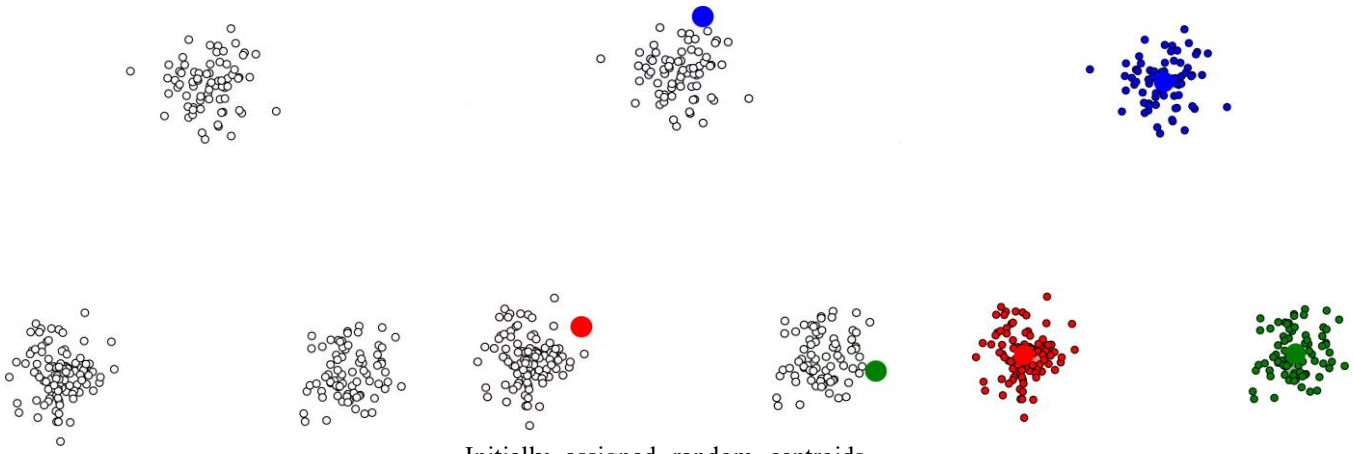
The k-means clustering is a simple algorithm that is used to group data points into a specified number of clusters. This popular machine learning algorithm is used in various applications such as computer vision. Algorithm 1 describes the steps constituting k-means clustering.

Algorithm 1: The k-means clustering algorithm

Data: Data points (D), Number of clusters (k)
Result: Cluster numbers (C) which specify the cluster number that each point belongs to

1. Randomly select k data points and set them as the k cluster centers (also called centroids);
2. **while** *there are changes in cluster numbers* C do
3. **foreach data point** in D **do**
4. Associate the data point with the nearest cluster; // Assigning a cluster to each point!
5. Recalculate the position of the k centroids using the mean value for each cluster of points.

Figure 1 shows how the k-means clustering algorithm groups a set of data points *D* into three clusters when $k = 3$ (for more visualizations, see this link). The clusters 0, 1, and 2 are indicated by the three colors red, green, and blue. The **output** of **our algorithm** will be an assignment of a cluster number to each data point. **Visualization is not required**.



The data points *D*

Initially assigned random centroids (shown as large red, green, and blue points)

The final positions of the centroids and clusters after the algorithm concluded. The three colours represent three clusters.

Figure 1: k-means clustering algorithm for a given set of data points *D* and number of clusters = 3

I. Grading Strategy:

The assignment must be submitted in the specified project structure format, packaged as a zip file. All code should be well-commented, clean, and stick to best coding practices. Each group must submit the assignment through the Canvas submission page, including all relevant files. You will be required to submit the code, run the Makefile, and execute the binary file. Below are the test scenarios that will be used to evaluate your submission. Ensure that you perform similar tests on your code before submitting it on Canvas.

Test Scenarios:

1. Correct Data Reading: Verify that the program reads the entire dataset from kmeans-data.txt correctly, accurately parsing the data points into a matrix.
2. Correct Clustering Output: Check whether the K-means algorithm correctly partitions the data into the specified number of clusters.
3. Edge Cases Handling: Test the program with edge cases, such as k = 1 and k = number of data points. The program should handle these without errors and produce meaningful results.
4. Error Handling: Run tests to ensure the program gracefully handles invalid inputs, such as non-numeric data, missing files, incorrect file formats, and any other possible errors. The program should provide appropriate and informative error messages for each type of issue encountered. It should not crash or produce unexpected behavior, and it should handle errors in a user-friendly manner to assist with debugging.
5. Performance Test: Assess the program's performance on the given dataset and a large 2D dataset to ensure it completes clustering within a reasonable time frame and without excessive memory usage. Test with different data sizes to confirm scalability and efficiency. **Include your additional test dataset(s)** in the submitted folder. Also, **include a tex**t file detailing the computer resources dedicated to your Unix environment. The aim is to evaluate your understanding of large datasets. In the text file, **briefly explain** why you consider your selected dataset(s) to be large.
6. Output Format: Verify that the output file **kmeans-output.txt** correctly contains the original data points with the assigned cluster number as the third column.
7. Memory Leakage and Debugging Checks: Perform memory leak detection using tools like Valgrind to ensure there are no memory leaks in the program. For debugging, you can use gdb to identify and resolve any runtime errors. Include these checks in the Makefile for automated testing. Proper memory management and debugging practices are essential **for achieving high grades**.
8. Code Quality: The code should be well-structured and to the best modular, following good programming practices. codes should be clearly defined. Comments should be included to explain key sections of the code.
9. Makefile: The Makefile should correctly compile the source files into object files and link them to produce the executable kmeans. It may include targets for cleaning, testing, and memory leak detection, which can contribute to achieving higher grades. The Makefile should be documented to describe its usage.
10. Execution and Output: Running ./kmeans in the project directory terminal should generate kmeans-output.txt with data points and their respective cluster assignments. Ensure the output file matches the expected format and includes all required information.
11. Environment: Programs must be written in C, compiled using gcc, and run in Unix environments, specifically Ubuntu 24.04 or Fedora 40. Ensure compatibility with these environments during development and testing.

Grading Criteria:

Grade: D

- All requirements from Section A to Section H (including all sections)
- Basic implementation of K-means clustering with dynamic memory allocation.
- Code compiles and runs without errors using the Makefile.
- Basic error handling is present (e.g., checking for file existence).
- The output file kmeans-output.txt is generated with clusters assigned but may not handle edge cases well.

Grade: C

- All requirements for Grade D.
- The program handles common edge cases, such as k = 1 or k = the number of data points.
- Proper memory management is implemented, with no memory leaks.
- The Makefile includes targets for compiling, linking, and cleaning object files, demonstrating a good understanding of program management.

Grade: B

- All requirements for Grade C.
- Robust error handling for various scenarios, including invalid input data, and providing useful error messages.
- Code is modular and well-organized, with clear separation between the main logic, K-means algorithm implementation with functions (e.g., for file I/O and memory management, etc.).
- Code quality is high, with detailed comments explaining key sections, especially around memory allocation, K-means steps, and error handling.

Grade: A

- All requirements for Grade B.
- Implementation of advanced features or optimizations, such as: Allowing user to select dataset, input for initial centroid selection, etc.
- Efficient handling of large datasets: Include your large test dataset(s) in the submission file, along with a text file detailing the computer resources used. In the text file, briefly explain why you consider your selected dataset(s) to be large.
- Exceptional error handling that anticipates a wide range of possible issues.
- Demonstrated use of debugging tools or techniques in Makefile (e.g., Valgrind for memory checks, gdb for runtime debugging).
- Code is extremely clean, well-documented, and demonstrates a clear understanding of the K-means algorithm, pointers, and memory management techniques, etc.
- The program and Makefile are designed with extensibility and maintainability in mind, showcasing an advanced level of code organization and project management skills.

Note: Even if your implementation fulfills the requirements for your target grade, poor code quality, insufficient commenting, or failure to adhere to the specified project structure can result in a reduction of one grade level. To ensure full credit, carefully follow all outlined requirements, maintain high standards in code quality, and rigorously test your program. Non-compliance with these standards, copying, using code generated by LLMs, or sharing the code may result in an F grade, with no feedback or opportunity for resubmission.

All the best!