Assignment 2: Hashing of Workplace Emails

This assignment aims to help you practice the **hash tables**, especially collision resolution with **open addressing**. Your main task in this assignment is to develop a hash table to store some of the emails you read in the first assignment using **C programming language**.

Overview

Let's say you want to keep the emails you read in the first assignment stored in a database, but you do not want them to be stored so blatantly in a normal table, especially when considering the performance advantages of hash tables when your system would hypothetically need to keep track of thousands of emails.

In this assignment, you will be developing a hash table with specific hashing techniques. The system shall be able to perform the following functionalities:

1. Read emails from files and put them in a hash table
2. Search for an email by taking ID and sender's name
3. Print table

Input

The program will have two inputs:

- Data files with each one storing an email. Each data file will be structured as following:

  Email_ID
  From: Sender
  To: Recipient
  Date: Day_of_the_month
  Content

  Each of these emails you read will be stored in a data structure defined as in the table below:

Information	Data Type
Email_ID	int
Sender	char[51]
Recipient	char[51]
Day_of_the_month	int
No_of_words	int

- Interactive input from console which will allow the user to read emails to the table based on the selected open addressing technique and decide what to be displayed next.

Hashing

The application shall create a hash table in which the user decides what open addressing technique to be used:

- If the user enters 1, then the table will use **double hashing (where f(i) = i * $hash_2$(key))**,
- But if they enter 2, then **linear probing (where f(i) = i)** will be used.

The initial size of the hash table should be 11. If the load factor λ (the total number of emails in a hash table / the size of the hash table) becomes larger than 0.5, then rehashing should be applied with the following steps:

1. Compute the size of the new hash table by multiplying the size of the old hash table by two and then rounding it to the next prime number
2. Dynamically allocate a new hash table and locate the emails into the new hash table properly
3. Destroy the old hash table

The following hash functions to be used:

- key = Email_ID + ASCII (sender[0]) - 65
- hash(key) = (key) mod hashTableSize
- hash2(key) = 5 – (key mod 5)

Process Model

You can find the process model illustrated in figure 1 below.

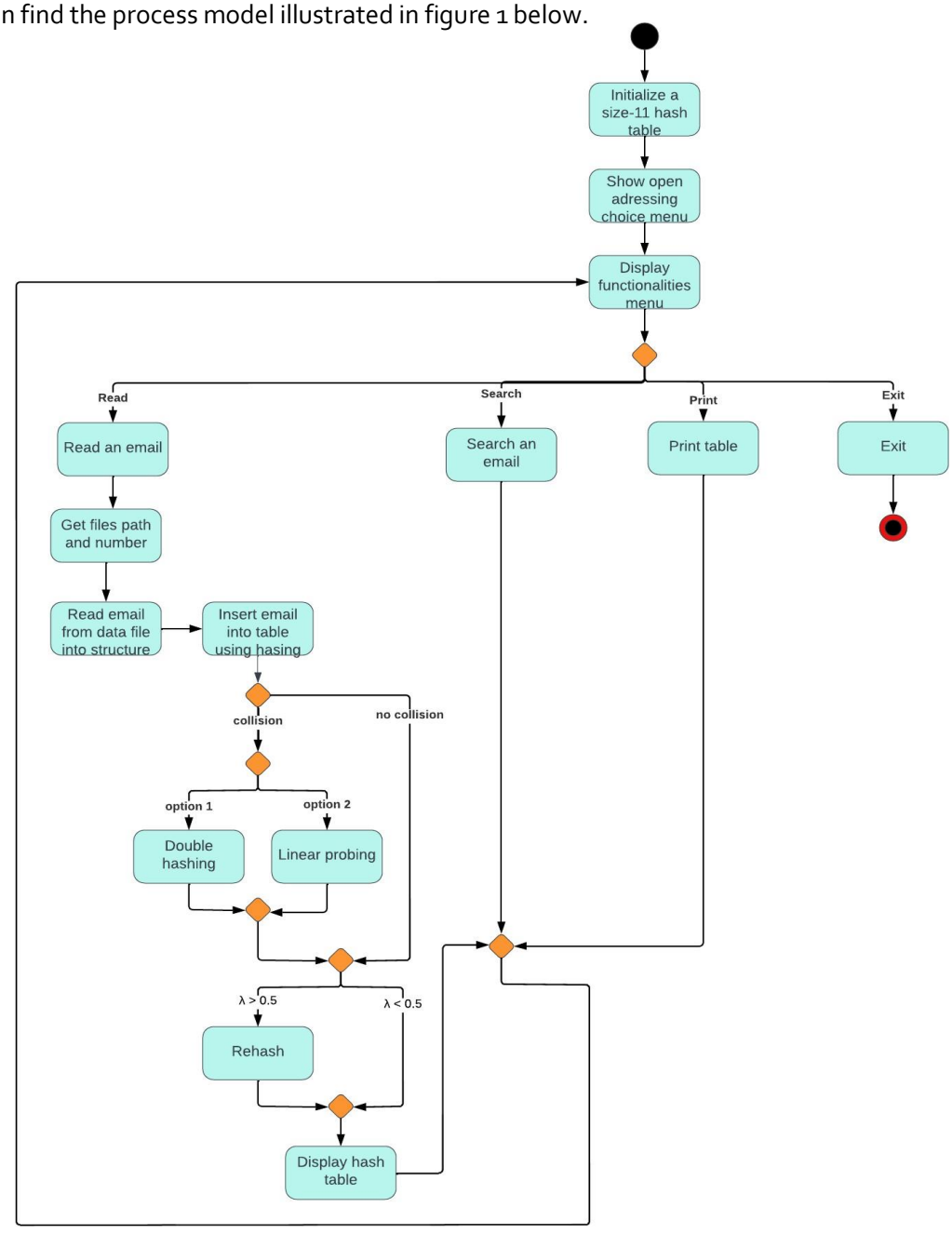

Figure 1: Process Model

First, the hash table's creation will take place, then user will pick whether **double hashing** or **linear probing** will be used to resolve collisions. After the choice is made, there is no going back on it unless the program is restarted. Then, the user will be presented with the following options:

1. **Read an email:**
   The operation will ask the user to enter a path to where email data files are stored and their number. Then, it will read the files which are named according to a number. For example, if there are 50 data files, then the files will be named "1.txt", "2.txt", all the way to "50.txt".

   Each time the function reads an email, it will insert it into the hash table and print the hash table after each insert. If an email is read from a file and the id for it already exists in the table, the program will skip reading the file while showing an appropriate error message.

   If the load factor of the hash table becomes more than 0.5, and then rehashing will be performed as mentioned above. Please note that when you do rehashing, you need to recompute the new positions for each element that was already in the table according to the new table size; if you just copy them into the same positions, you will lose a lot of points in this part.

2. **Search an email:**
   The operation will ask for **an email id and a sender's name** and then try to find the email in the hash table; it prints the email's details if found. An example is provided below where the input is shown in bold. If the student is not found, then the application should print "Student is not found!" and go back to the main menu. This operation should be executed using hashing functions, not a simple linear search. You will not get any points if you just put a simple sequential or linear search.

   ```
   Enter unique identifier: 52
   Enter sender: Perla
   Recipient: Ash
   Date: 9
   Number of words: 11
   ```

3. **Print Table:**
   This function will simply print the contents of the has table in the order they are placed. As seen in the output below, if there is no element at a certain index, then the function will show nothing. You can find below examples of the output when either double hashing or linear probing are used. This output shows the table after inserting the emails in the text files 1.txt to 5.txt in order. These text files are attached with the assignment.

   Output with **Linear Probing**:

   ```
   Index      ID         Sender     Recipient    Date     Words
   0
   1          52         Perla      Ash          9        11
   2          2          Ash        Molly        22       5
   3          3          Karen      John         17       14
   4          11         Melina     Perla        5        12
   5
   6
   7
   8
   9
   10         1          John        Molly      16        8
   ```

Output with **Double Hashing**:

```
Index       ID          Sender      Recipient     Date      Words
0
1           52          Perla       Ash           9         11
2           2           Ash         Molly         22        5
3           11          Melina      Perla         5         12
4           3           Karen       John          17        14
5
6
7
8
9
10          1           John         Molly       16         8
```

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow modular programming which means that you are expected to divide your program into a set of meaningful functions.

Rules

- You need to write your program by using C programming.
- Code quality, modularity, efficiency, maintainability, and appropriate comments will be part of the grading.
- **You need strictly follow the specifications given in this document.**
- **It is suggested to use helper functions to support modular development.**

Grading Scheme

Grading Item	**Mark (out of 100)**
Email data structure and its correct use in the program	5
Main function where the menu operations are performed, and appropriate functions are called.	10
Read emails from the files and add them to a hash table	20
Double hashing	10
Linear probing	10
Rehashing	25
Search email	15
Print table	5