Assignment 1 Part 2 of 3

Launching into Computer Science

Word count: 833

README

Customer Identity Book

This Python script implements a virtual Customer Identity Book (CIB) using a console-based menu that determines the action based on the user's choice. The system permits users to perform various operations, add and remove customers, search for customers, update customer details, see all customers' details, and sort customers according to their last names.

Customer data are saved only during the program runtime. Customers are not saved externally once the termination has been completed.

Code Structure

The structure of the CIB code consists from the several main parts:

- **Functions:**
    - **‘display_menu()’** - function to display the menu.

```
# Function to display the menu
def display_menu():
    print("\nMain Menu\n")
    print("1 ) Add Customers")
    print("2 ) Remove Customers")
    print("3 ) Search Customers")
    print("4 ) Update Customer Details")
    print("5 ) Show All Customers Details")
    print("6 ) Sort Customers (Ascending)")
    print("7 ) Sort Customers (Descending)")
    print("0 ) Quit")
```

    - **‘is_customers_empty()’** - checks if the list of customers is empty, and returning **True** if the list is empty, **False** otherwise. This function developed on the second stage.

```
# Function to check if customers list is empty
def is_customers_empty():
    return not customers
```

    - **‘sort_ascending()’** and **‘sort_descending()’** - implements the bubble sort algorithm to sort a list of customers based on their **‘Last_name’** attribute in ascending and descending order, respectively. At the first stage it was planned one sorting function, however, changes applied to provide ascending and descending order sorting at the second stage. Although it is not the most efficient sorting algorithm for large datasets, however, it is understandable and easy to implement (Lopez, 2022).

```
# Function to sort customers in ascending order based on Last_name
def sort_ascending(customers):
    # Get the number of customers in the list
    n = len(customers)
    # Outer loop: Iterate through all elements in the list
    for i in range(n - 1):
        # Inner loop: Iterate through the unsorted part of the list
        for j in range(0, n - i - 1):
            # Compare Last_Name for adjacent customers and swap if
            necessary
            if customers[j]["Last_name"] > customers[j +
            1]["Last_name"]:
                # Swap the positions of the customers
                customers[j], customers[j + 1] = customers[j + 1],
                customers[j]
```

```
# Function to sort customers in descending order based on Last_name
def sort_descending(customers):
    # Get the number of customers in the list
    n = len(customers)
    # Outer loop: Iterate through all elements in the list
    for i in range(n - 1):
        # Inner loop: Iterate through the unsorted part of the list
        for j in range(0, n - i - 1):
            # Compare Last_Name for adjacent customers and swap if
            necessary
            if customers[j]["Last_name"] < customers[j +
            1]["Last_name"]:
                # Swap the positions of the customers
                customers[j], customers[j + 1] = customers[j + 1],
                customers[j]
```

- **‘validate_email()’** - function to validate an email input.

```
# Function to validate email
def validate_email(email):
    # Check if the email has the correct structure
    if '@' in email and '.' in email:
        # Split the email into local part and domain part
        local_part, domain_part = email.split('@')
        # Check if the local and domain parts are not empty
        if local_part and domain_part:
            # Check if the domain part has at least one dot (.)
```

```
            if '.' in domain_part:
                return True
    # If any condition fails, the email is considered invalid
    return False
```

- **‘validate_phone_number()’** - function to validate a phone number input.

```
# Function to validate phone number
def validate_phone_number(phone_number):
    # Check if the phone number is composed only of digits and has a
length of 9
    return phone_number.isdigit() and len(phone_number) == 9
```

- **‘validate_birth_date()’** - function to validate a birth date input.

```
# Function to validate birth date
def validate_birth_date(birth_date):
    # Check if the birth date is composed only of digits and has a
length of 8
    return birth_date.isdigit() and len(birth_date) == 8
```

- **‘validate_id_number ()’** - function to validate an ID number input.

```
# Function to validate ID number
def validate_id_number(id_number):
    # Check if the ID number is composed only of digits and has a length
    of 6
    return id_number.isdigit() and len(id_number) == 6
```

- **Customer Object:**
  - **Customer ObjectDefinition:** a dictionary named ‘**`Customer`**‘ is defined with properties such as `Last_name`, `First_name`, `Birth_date`, `Address`, `Id_number`, `E_mail`, and `Phone_number`.

```
# Define the Customer object with properties
Customer = {
    "Last_name": None,
    "First_name": None,
    "Birth_date": None,
    "Address": None,
    "Id_number": None,
    "E_mail": None,
    "Phone_number": None
}
```

- **Initialization:** an empty list named '**`customers`**' is created to store customer objects.

```
# Initialize an empty list to store customer objects
customers = []
```

- **Data Collection:**
  - **Data Collection:** a loop is used to gather customer information from user input. It iterates a number of times specified by the variable '**`num`**', according to the assignment's requirements.

```
    # Prompt user for customer information
    customer["Last_name"] = input("Last Name: ")
    customer["First_name"] = input("First Name: ")
    customer["Birth_date"] = int(input("Birth Date: "))
    customer["Address"] = input("Address: ")
    customer["Id_number"] = int(input("ID Number: "))
    customer["E_mail"] = input("E-mail: ")
    customer["Phone_number"] = int(input("Phone Number: "))
    # Append the customer object to the list
    customers.append(customer)
```

  - **Display Collected Data:** after data collection, the program displays a header for the collected data and then prints each customer's details in tabular format using a loop.

```
    # Show collected customers details
    # Display header for the collected data
    print("\nID\t\tName\t\t\tBirth date\t\t\tAddress\t\t\tID Number\t\t\tE-
    mail\t\t\tPhone Number\n")
    # Loop to print customer details
    for idx, customer in enumerate(customers, 1):
        print("{}\t\t{} {}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}".format(
            idx, customer["Last_name"], customer["First_name"],
            customer["Birth_date"], customer["Address"],
            customer["Id_number"], customer["E_mail"],
            customer["Phone_number"]
        ))
```

- **Main Loop:** the main loop for the code execution, where the user interacts with the menu and performs operations (details are provided in the section – Application Features).

Application Features

The function begins with a while **`True`** loop that prompts a user's choice of action based on selection statements. After every iteration, the user is presented with a menu to select an option. Depending on the user's choice, a program performs various operations.

```
while True:
    # Display the menu
     display_menu()
     # Prompt user for menu choice
     choice = input("\nEnter your choice (0-7): ")
```

The main features of the script responsible for overall functioning are as follows:

Add New Customers: the function asks the user to enter the customer's details (as shown below). After hitting enter, the contact is saved and the confirmation prompt prints.

```
# User input to add customer
if choice == '1':
    # Add Customers
    # Create a copy of the Customer template to store user input
    customer = Customer.copy()
    # Prompt the user to enter the following details
    print("\nPlease enter the following details: ")
    # Collect user input for customer details
    customer["Last_name"] = input("Last Name: ")
    customer["First_name"] = input("First Name: ")
    customer["Birth_date"] = int(input("Birth Date: "))
    customer["Address"] = input("Address: ")
    customer["Id_number"] = int(input("ID Number: "))
    customer["E_mail"] = input("E-mail: ")
    customer["Phone_number"] = int(input("Phone Number: "))
    # Append the customer details to the list of customers
    customers.append(customer)
    # Display a success message with a check mark symbol
    print("\nCustomer details saved successfully \N{check mark}")
```

All the features below apply the function that first checks whether the **`customers`** is empty or not, if **`True`**, the user is asked to add a new contact and try again.

Remove Customers: user can remove customers based on their **`Last_name`** from the customer list, it deletes the key:value pair of that particular **`customer`** from **`customers`**.

```
# User input to remove customer
elif choice == '2':
    # Calls function to check if customers list is empty
    if is_customers_empty():
        # Display a message if the customers list is empty
        print("No customers to remove. Please add a new customer and
        try again.")
    else:
        # Remove Customers
        # Prompt the user to enter the Last Name of the customer to
        remove
        remove_last_name = input("Enter the Last Name of the customer
        to remove: ")
        # Create a list of customers with matching Last Names (case-
        insensitive)
        removed_contacts = [customer for customer in customers if
        customer["Last_name"].lower() == remove_last_name.lower()]
```

```
        # Check if there are matching customers to remove
        if removed_contacts:
            # Iterate through the list of matching customers and remove
            them
            for removed_contact in removed_contacts:
                customers.remove(removed_contact)
            # Display a success message
            print("Customers with Last Name '{}' removed
            successfully.".format(remove_last_name))
        else:
            # Display a message if no customers were found with the
            specified Last Name
            print("No customers found with Last Name
            '{}'.".format(remove_last_name))
```

Search Customers: user can search customers based on their **`Last_name`** from the customer list by iterating through each customer in the list to see if the input is a part of the customer's last name. If a match is found, it displays the details relevant to the customer such as the full name and ID number. Otherwise, it notifies the user that the name is not found.

```
# User input to search customer by the Last Name
elif choice == '3':
    # Calls function to check if customers list is empty
    if is_customers_empty():
        # Inform the user if the customers list is empty
        print("No customers to search. Please add a new customer and
        try again.")
    else:
        # Search Customers
        # Prompt user to enter the customer's Last Name for search
        search_term = input("\nEnter customer Last Name: ")
        # Displaying the search result header
        print("Search result:")
        # Initializes a variable to keep track of whether a match is
        found during the search
        found = False
        # Iterate through each customer to find a match
        for customer in customers:
            # Check if the search term is present in either First Name
            or Last Name (case-insensitive)
```

```
            if search_term.lower() in customer["Last_name"].lower():
                # Display relevant details for the found customer
                print("Name: {} {}, ID Number:
                {}".format(customer["First_name"],
                customer["Last_name"], customer["Id_number"]))
                # Indicate that a match has been found
                found = True
        # Inform the user if no match is found
        if not found:
            print("Name not found")
```

Update Customers Details: user can update customers based on their **`Last_name`** from the customer list. The program prints customer details and asks to update information. Otherwise prints error message.

```
# User input to update customer details
elif choice == '4':
    # Calls function to check if customers list is empty
    if is_customers_empty():
        print("No customers to update. Please add a new customer and
        try again.")
    else:
        # Update Customer Details
        # Prompt user to enter the Last Name of the customer to update
        update_last_name = input("\nEnter the Last Name of the customer
        to update: ")
        # Create a list of customers matching the entered Last Name
        update_customers = [customer for customer in customers if
        customer["Last_name"].lower() == update_last_name.lower()]
        if update_customers:
            # Display customer details for the user to choose from
            for customer in update_customers:
                print("\nUpdate Customer Details for:\n")
                # Display header for customer details
                print("\nID\t\t\tName\t\t\tBirth
                date\t\t\tAddress\t\t\tID Number\t\t\tE-mail\t\t\tPhone
                Number\n")
                # Display current customer details
                print("{}\t\t\t{}
                {}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}".format(
                    customers.index(customer) + 1,
                    customer["Last_name"], customer["First_name"],
                    customer["Birth_date"], customer["Address"],
                    customer["Id_number"], customer["E_mail"],
```

```
                    customer["Phone_number"]
                ))
                # Prompt user for updated information
                print("\nPlease enter the following details to update:
                ")
                customer["Last_name"] = input("Last Name: ")
                customer["First_name"] = input("First Name: ")
                customer["Birth_date"] = int(input("Birth Date: "))
                customer["Address"] = input("Address: ")
                customer["Id_number"] = int(input("ID Number: "))
                customer["E_mail"] = input("E-mail: ")
                customer["Phone_number"] = int(input("Phone Number: "))
                # Display a success message
                print("\nCustomer details updated successfully.")
        else:
            # Inform the user if no customers are found with the
            entered Last Name
            print("\nNo customers found with Last Name
            '{}'.".format(update_last_name))
```

Show All Customers: user can view and display all saved customers.

```
# User input to show all customers details
elif choice == '5':
    # Calls function to check if customers list is empty
    if is_customers_empty():
        print("No customers to show. Please add a new customer and try
        again.")
    else:
        # Show all customers details
        print("\nID\t\t\tName\t\t\tBirth date\t\t\tAddress\t\t\tID
        Number\t\t\tE-mail\t\t\tPhone Number\n")
    # Iterate through each customer and display their details
    for idx, customer in enumerate(customers, 1):
        print("{}\t\t\t{}
        {}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}".format(idx,
        customer["Last_name"], customer["First_name"],
        customer["Birth_date"], customer["Address"],
        customer["Id_number"], customer["E_mail"],
        customer["Phone_number"]
    ))
```

Sort Customers: user can sort a list of customers in ascending and descending order based on their **`Last_name`**, calling functions. Both sorting options provided as a two separate features (No. 6 and 7).

```
# User input to sort customers (ascending)
elif choice == '6':
    # Calls function to check if customers list is empty
    if is_customers_empty():
        print("No customers to show. Please add a new customer and try
        again.")
    else:
        # Sort customers details based on Last_Name (ascending)
        sort_ascending(customers)
        # Print success message after sorting
        print("\nCustomers sorted successfully based on Last_Name
        (ascending).")
        # Print header for the sorted customer details
        print("\nID\t\tName\t\t\tBirth date\t\t\tAddress\t\t\tID
        Number\t\t\tE-mail\t\t\tPhone Number\n")
        # Iterate through sorted customers and print their details
        for idx, customer in enumerate(customers, 1):
            print("{}\t\t{}
            {}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}".format(
                idx, customer["Last_name"], customer["First_name"],
                customer["Birth_date"], customer["Address"],
                customer["Id_number"], customer["E_mail"],
                customer["Phone_number"]
        ))
```

Descending sorting option:

```
# User input to sort customers (descending)
elif choice == '7':
            # Calls function to check if customers list is empty
    if is_customers_empty():
        print("No customers to show. Please add a new customer and try
        again.")
    else:
        # Sort customers details based on Last_Name (descending)
        sort_descending(customers)
        # Print success message after sorting
        print("\nCustomers sorted successfully based on Last_Name
        (descending).")
        # Print header for the sorted customer details
        print("\nID\t\tName\t\t\tBirth date\t\t\tAddress\t\t\tID
        Number\t\t\tE-mail\t\t\tPhone Number\n")
        # Iterate through sorted customers and print their details
        for idx, customer in enumerate(customers, 1):
            print("{}\t\t{}
            {}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}\t\t\t{}".format(
```

```
                idx, customer["Last_name"], customer["First_name"],
                customer["Birth_date"], customer["Address"],
                customer["Id_number"], customer["E_mail"],
                customer["Phone_number"]
            ))
```

Quit:

The software ends the program's execution by breaking out of the loop and printing a message to the user informing them that the programme has ended.

```
# User input to exit the program
elif choice == '0':
    # Quit message
    print("Program Exited\n")
    # Exit the loop to end the program
    Break
```

Validation

This code validates the accuracy of the data entered by the user. These validations were implemented while collecting and updating customer information to ensure the accuracy of the stored data. If none of the validation conditions are met, users are prompted to re-enter the information. Implementation examples of email, phone number, birth date, and ID number validations are provided in the main loop:

```
# Create a new customer object
customer = Customer.copy()
# Prompt user for customer information
print("\nPlease enter the following details: ")
customer["Last_name"] = input("Last Name: ")
customer["First_name"] = input("First Name: ")
# Validate birth date format using the validate_birth_date function
birth_date = input("Birth Date (YYYYMMDD): ")
while not validate_birth_date(birth_date):
    print("Invalid birth date format. Please enter YYYYMMDD.")
    birth_date = input("Birth Date (YYYYMMDD): ")
customer["Birth_date"] = int(birth_date)
```

```
    customer["Address"] = input("Address: ")
    # Validate ID Number format using the validate_id_number function
    id_number = input("ID Number (min 6-digits): ")
    while not validate_id_number(id_number):
        print("Invalid ID Number format. Please enter min 6 digits.")
        id_number = input("ID Number (min 6-digits): ")
    customer["Id_number"] = int(id_number)
    # Validate email format using the validate_email function
    email = input("Email: ")
    while not validate_email(email):
        email = input("Invalid email format. Please enter a valid email
        address: ")
    customer["E_mail"] = email
    # Validate phone number format using the is_valid_phone_number function
    phone_number = input("Phone Number (9-digit): ")
    while not validate_phone_number(phone_number):
        print("Invalid phone number format. Please enter a valid 9-digit
        phone number.")
        phone_number = input("Phone Number (9-digit): ")
    customer["Phone_number"] = int(phone_number)
    # Append the customer object to the list
    customers.append(customer)
```

However, the implementation of advanced email validation using error messages in the main loop was not successful:

```
# Function to validate email
def validate_email(email):
    # Check if '@' is present in the email
    if "@" not in email:
        print("Invalid email: Missing '@'")
        return False
    # Check if valid_extensions is not empty
    valid_extensions = [".com", ".org", ".edu", ".gov", ".net"]
    if not valid_extensions:
        print("Error: No valid domain extensions defined")
        return False
    # Check if the email ends with a valid domain extension
    if not any(email.endswith(ext) for ext in valid_extensions):
        print("Invalid email: Invalid domain extension")
        return False
    # If all conditions pass, the email is considered valid
    return True
```

```
# Example of usage in main loop:
# Validate email format using the validate_email function
email = input("Email: ")
email_error = validate_email(email)
while email_error:
    print(email_error)
    email = input("Email: ")
    email_error = validate_email(email)
customer["E_mail"] = email
```

Conclusion

The described code illustrates a customer management system with an interface based on menu selection. It allows users to perform various operations, such as adding customers, removing customers, searching for customers, updating customer details, showing all customer details, and sorting customers according to their last name in ascending and descending order. Input validation functions that verify whether the information provided by the user, such as email addresses, phone numbers, birth dates, and ID numbers, meets certain criteria are implemented in the system.

Reference list

Lopez, E. (2022) *Bubble sort - how it works, Psuedocode and C++ & Python implementation*, *Medium*. Available at: https://medium.com/codex/bubble-sort-how-it-works-psuedocode-and-c-python-implementation-c45306d44827 [Accessed: 27 January 2024].

Testing Strategy

Testing plays a crucial role in the software development process to ensure that the application works properly and meets the requirements, which was set in Assignment 1 Part 1 of 3 - Data Structures and Algorithm Design (see xx page). The testing strategy for the CIB code is given below.

Table 1. CIB application test summary

No	Scenario	Comment	Test status
1	Add 5 customer entries according to the requirements		Passed
2	**Add Customers** **Objective:** Verify that customers can be successfully added.		Passed
3	**Remove Customers** **Objective:** Verify that customers can be successfully removed.		Passed
4	**Search Customers** **Objective:** Verify that the program can successfully search for customers		Passed
5	**Update Customer Details** **Objective:** Verify that customer details can be successfully updated.		Passed
6	**Show All Customers Details** **Objective:** Verify that the program can display all customer details.		Passed
7	**Sort Customers Details Ascending** **Objective:** Verify that the program can sort all customer details based on last name.	Updated test scenario content	Passed
8	**Sort Customers Details Descending** **Objective:** Verify that the program can sort all customer details based on last name.	Added new test scenario	Passed
9	**Menu Choices** **Objective:** Ensure that the program handles invalid menu choices gracefully.	Updated test scenario content	Passed
10	**Numeric Input for Birth Date and ID Number** **Objective:** Verify that the program handles numeric input for birth date and ID number.		Passed
11	**Numeric Input for Phone Number** **Objective:** Verify that the program handles numeric input for phone number.		Passed
12	**Valid Input for Email** **Objective:** Verify that the program handles valid input for email.		Passed
13	**Valid If Customer List is not Empty** **Objective:** Verify that the program valid customer list.	Added new test scenario	Passed

Table 2. CIB application test results

Scenario	Test case	Expected Outcome
Customer Management Operations		
Add 5 customer entries according to the requirements	Customer = { "Last_name": "Doe", "First_name": "Thomas", "Birth_date": 19900101, "Address": "Oak 1, Vilnius, LT", "Id_number": 111111, "E_mail": "thomas.doe@email.com", "Phone_number": 511511511 "Last_name": "Nuu", "First_name": "Mark", "Birth_date": 19900202, "Address": "Oak 2, Vilnius, LT", "Id_number": 222222, "E_mail": "mark.nuu@email.com", "Phone_number": 512512512 "Last_name": "Wue", "First_name": "David", "Birth_date": 19900303, "Address": "Oak 3, Vilnius, LT", "Id_number": 333333, "E_mail": "david.wue.doe@email.com", "Phone_number": 513513513 "Last_name": "Liu", "First_name": "Joe", "Birth_date": 19900404, "Address": "Oak 4, Vilnius, LT", "Id_number": 444444, "E_mail": "joe.liu@email.com", "Phone_number": 514514514	The program should successfully add customers to the list from the beginning. ID Name Birth date Address 1 Doe Thomas 19900101 Oak 1, Vilnius, LT 2 Nuu Mark 19900202 Oak 2, Vilnius, LT 3 Wue David 19900303 Oak 3, Vilnius, LT 4 Liu Joe 19900404 Oak 4, Vilnius, LT 5 Key Sven 19900505 Oak 5, Vilnius, LT ID Number E-mail Phone Number 111111 thomas.doe@email.com 511511511 222222 mark.nuu@email.com 512512512 333333 david.wue@email.com 513513513 444444 joe.liu@email.com 514514514 555555 sven.key@email.com 515515515

	"Last_name": "Key", "First_name": "Sven", "Birth_date": 19900505, "Address": "Oak 5, Vilnius, LT", "Id_number": 555555, "E_mail": "sven.key@email.com", "Phone_number": 515515515 }	
Console output:		

```
PS C:\Users\pc> & C:/Users/pc/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pc/Desktop/Phone book/cib.py"

Please enter the following details:
Last Name: Doe
First Name: Thomas
Birth Date (YYYYMMDD): 19900101
Address: Oak 1, Vilnius, LT
ID Number (min 6-digits): 111111
Email: thomas.doe@email.com
Phone Number (9-digit): 511511511

Please enter the following details:
Last Name: Nuu
First Name: Mark
Birth Date (YYYYMMDD): 19900202
Address: Oak 2, Vilnius, LT
ID Number (min 6-digits): 222222
Email: mark.nuu@email.com
Phone Number (9-digit): 512512512

Please enter the following details:
Last Name: Wue
First Name: David
Birth Date (YYYYMMDD): 19900303
Address: Oak 3, Vilnius, LT
ID Number (min 6-digits): 333333
Email: david.wue@email.com
Phone Number (9-digit): 513513513

Please enter the following details:
Last Name: Liu
First Name: Joe
Birth Date (YYYYMMDD): 19900404
Address: Oak 4, Vilnius, LT
ID Number (min 6-digits): 444444
Email: joe.liu@email.com
Phone Number (9-digit): 514514514
Address: Oak 4, Vilnius, LT
ID Number (min 6-digits): 444444
Email: joe.liu@email.com
Phone Number (9-digit): 514514514

Please enter the following details:
Last Name: Key
First Name: Sven
Birth Date (YYYYMMDD): 19900505
Address: Oak 5, Vilnius, LT
ID Number (min 6-digits): 555555
Email: sven.key@email.com
Phone Number (9-digit): 515515515

ID              Name                    Birth date                      Address                 ID Number                       E-mail                  Phone Number

1               Doe Thomas                      19900101                        Oak 1, Vilnius, LT                      111111                  thomas.doe@email.com                    511511511
2               Nuu Mark                        19900202                        Oak 2, Vilnius, LT                      222222                  mark.nuu@email.com                      512512512
3               Wue David                       19900303                        Oak 3, Vilnius, LT                      333333                  david.wue@email.com                     513513513
4               Liu Joe                 19900404                        Oak 4, Vilnius, LT                      444444                  joe.liu@email.com                       514514514
5               Key Sven                        19900505                        Oak 5, Vilnius, LT                      555555                  sven.key@email.com                      515515515

Main Menu

1 ) Add Customers
2 ) Remove Customers
3 ) Search Customers
4 ) Update Customer Details
5 ) Show All Customers Details
6 ) Sort Customers (Ascending)
7 ) Sort Customers (Descending)
0 ) Quit

Enter your choice (0-7):
```

Add Customers **Objective:** Verify that customers can be successfully added.	1. Choose option '1' to add customers. 2. Enter information for multiple customers: Customer = { "Last_name": "Boo", "First_name": "Drake", "Birth_date": 19900606, "Address": "Oak 6, Vilnius, LT", "Id_number": 666666, "E_mail": "drake.boo@email.com", "Phone_number": 516516516 } 3. Verify that the customers are added to the list.	The program should successfully add customers to the list. ID Name Birth date Address 6 Boo Drake 19900606 Oak 6, Vilnius, LT ID Number E-mail Phone Number 666666 drake.boo@email.com 516516516
Console output: Enter your choice (0-7): 1 Please enter the following details: Last Name: Boo First Name: Drake Birth Date (YYYYMMDD): 19900606 Address: Oak 6, Vilnius, LT ID Number (min 6-digits): 666666 Email: drake.boo@email.com Phone Number (9-digit): 516516516 Phone Number (9-digit): 516516516 Customer details saved successfully ✓		
Remove Customers **Objective:** Verify that customers can be successfully removed.	1. Choose option '2' to remove customers. 2. Enter the last name of an existing customer. "Last_name": "Liu" 3. Verify that the customer is removed from the list.	The program should successfully remove the specified customer.
Console output:		

`` Enter your choice (0-7): 2 Enter the Last Name of the customer to remove: Liu Customers with Last Name 'Liu' removed successfully. ```		
Search Customers **Objective:** Verify that the program can successfully search for customers	1. Choose option '3' to search for customers. 2. Enter a search term (last name). "Last_name": "Nuu" 3. Verify that the program displays the relevant customer information.	The program should successfully display customer information based on the search term. Name: Mark Nuu, ID Number: 222222
Console output: ``` Enter your choice (0-7): 3 Enter customer Last Name: Nuu Search result: Name: Mark Nuu, ID Number: 222222 ```		
Update Customer Details **Objective:** Verify that customer details can be successfully updated.	1. Choose option '4' to update customer details. 2. Enter the last name of an existing customer. "Last_name": "Doe" 3. Verify that the program displays the customer details for confirmation. 4. Enter updated information. 5. Verify that the customer details are updated.	The program should successfully update the specified customer's details.
Console output:		

```
Enter your choice (0-7): 4

Enter the Last Name of the customer to update: Doe

Update Customer Details for:

ID                  Name                Birth date          Address             ID Number                   E-mail              Phone Number

1             Doe Thomas                19900101            Oak 1, Vilnius, LT                  111111              thomas.doe@email.com                 511511511

Please enter the following details to update:
Last Name: Doe
First Name: Thomas
Birth Date (YYYYMMDD): 19900101
Address: Newstreet 5, Vilnius, LT
ID Number (min 6-digits): 111111
Email: t.doe@email.com
Phone Number (9-digit): 611611611
Phone Number (9-digit): 611611611

Customer details updated successfully.
```

Show All Customers Details **Objective:** Verify that the program can display all customer details.	1. Choose option '5' to show all customer details. 2. Verify that the program displays a formatted list of all customers.	The program should successfully display all customer details. ID Name Birth date Address 1 Doe Thomas 19900101 Oak 1, Vilnius, LT 2 Wue David 19900303 Oak 3, Vilnius, LT 3 Liu Joe 19900404 Oak 4, Vilnius, LT 4 Key Sven 19900505 Oak 5, Vilnius, LT 5 Boo Drake 19900606 Oak 6, Vilnius, LT ID Number E-mail Phone Number 111111 thomas.doe@email.com 511511511 333333 david.wue@email.com 513513513 444444 joe.liu@email.com 514514514 555555 sven.key@email.com 515515515 666666 drake.boo@email.com 516516516

Console output:

```
Enter your choice (0-7): 5

ID          Name              Birth date          Address                ID Number          E-mail              Phone Number

1           Doe Thomas              19900101              Newstreet 5, Vilnius, LT              111111              t.doe@email.com              611611611
2           Nuu Mark                19900202              Oak 2, Vilnius, LT          222222              mark.nuu@email.com           512512512
3           Wue David               19900303              Oak 3, Vilnius, LT          333333              david.wue@email.com          513513513
4           Key Sven                19900505              Oak 5, Vilnius, LT          555555              sven.key@email.com           515515515
5           Boo Drake               19900606              Oak 6, Vilnius, LT          666666              drake.boo@email.com          516516516
```

Sort Customers Details Ascending **Objective:** Verify that the program can sort all customer details based on last name.	1. Choose option '6' to sort all customer details. 2. Verify that the program prints a success message. 3. Verify that the customer details are displayed in sorted order.	The program should successfully sort and display all customer details based on last name. ID Name Birth date Address 1 Boo Drake 19900606 Oak 6, Vilnius, LT 2 Doe Thomas 19900101 Oak 1, Vilnius, LT 3 Key Sven 19900505 Oak 5, Vilnius, LT 4 Liu Joe 19900404 Oak 4, Vilnius, LT 5 Wue David 19900303 Oak 3, Vilnius, LT ID Number E-mail Phone Number 666666 drake.boo@email.com 516516516 111111 thomas.doe@email.com 511511511 555555 sven.key@email.com 515515515 444444 joe.liu@email.com 514514514 333333 david.wue@email.com 513513513
Console output:		

```
Enter your choice (0-7): 6

Customers sorted successfully based on Last_Name (ascending).

ID          Name                Birth date          Address                ID Number          E-mail                Phone Number

1           Boo Drake                 19900606                Oak 6, Vilnius, LT          666666                drake.boo@email.com          516516516
2           Doe Thomas                19900101                Newstreet 5, Vilnius, LT           111111                t.doe@email.com          611611611
3           Key Sven                  19900505                Oak 5, Vilnius, LT          555555                sven.key@email.com           515515515
4           Nuu Mark                  19900202                Oak 2, Vilnius, LT          222222                mark.nuu@email.com           512512512
5           Wue David                 19900303                Oak 3, Vilnius, LT          333333                david.wue@email.com          513513513
```

Sort Customers Details Descending **Objective:** Verify that the program can sort all customer details based on last name.	1. Choose option '7' to sort all customer details. 2. Verify that the program prints a success message. 3. Verify that the customer details are displayed in sorted order.	The program should successfully sort and display all customer details based on last name. ID Name Birth date Address 5 Wue David 19900303 Oak 3, Vilnius, LT 4 Liu Joe 19900404 Oak 4, Vilnius, LT 3 Key Sven 19900505 Oak 5, Vilnius, LT 2 Doe Thomas 19900101 Oak 1, Vilnius, LT 1 Boo Drake 19900606 Oak 6, Vilnius, LT ID Number E-mail Phone Number 333333 david.wue@email.com 513513513 444444 joe.liu@email.com 514514514 555555 sven.key@email.com 515515515 111111 thomas.doe@email.com 511511511 666666 drake.boo@email.com 516516516
Console output:		

```
Enter your choice (0-7): 7

Customers sorted successfully based on Last_Name (descending).

ID          Name                Birth date          Address                 ID Number           E-mail              Phone Number

1           Wue David           19900303            Oak 3, Vilnius, LT      333333                  david.wue@email.com          51351351
2           Nuu Mark            19900202            Oak 2, Vilnius, LT      222222                  mark.nuu@email.com           51251251
3           Key Sven            19900505            Oak 5, Vilnius, LT      555555                  sven.key@email.com           51551551
4           Doe Thomas          19900101            Newstreet 5, Vilnius, LT            111111          t.doe@email.com          61161161
5           Boo Drake           19900606            Oak 6, Vilnius, LT      666666                  drake.boo@email.com          51651651
```

User Input Validation:

Menu Choices **Objective:** Ensure that the program handles invalid menu choices gracefully.	1. Enter an invalid menu choice (e.g., '8'). 2. Verify that the program prints an error message. 3. Repeat the test with other invalid choices.	The program should inform the user of the invalid choice.

Console output:

```
Enter your choice (0-7): 8
Invalid choice. Please enter a number between 0 and 7.
```

Numeric Input for Birth Date and ID Number **Objective:** Verify that the program handles numeric input for birth date and ID number.	1. Enter non-numeric characters for birth date or ID number. 2. Verify that the program prompts the user for valid numeric input. 3. Enter valid numeric input for birth date and ID number.	The program should handle non-numeric input gracefully and request valid numeric input.

Console output:

```
Birth Date (YYYYMMDD): 1978
Invalid birth date format. Please enter YYYYMMDD.
Birth Date (YYYYMMDD):
```

Numeric Input for Phone Number **Objective:** Verify that the program handles numeric input for phone number.	1. Enter non-numeric characters for phone number. 2. Verify that the program prompts the user for valid numeric input. 3. Enter valid numeric input for phone number.	The program should handle non-numeric input gracefully and request valid numeric input.

Console output:

```
ID Number (min 6-digits): 111
Invalid ID Number format. Please enter min 6 digits.
ID Number (min 6-digits):
```

```
Phone Number (9-digit): 123
Invalid phone number format. Please enter a valid 9-digit phone number.
Phone Number (9-digit):
```

Valid Input for Email **Objective:** Verify that the program handles valid input for email.	1. Enter email without “@” symbol. 2. Verify that the program prompts the user for valid email input. 3. Enter valid format input for email.	The program should handle non-valid input gracefully and request valid email input

Console output:

```
Email: a.a.com
Invalid email format. Please enter a valid email address:
```

Valid If Customer List is not Empty **Objective:** Verify that the program valid customer list.	1. Delete all customers. 2. Choose any option from menu (except No.1 and 0).	The program should inform user that customer list is empty and program asks to add new customer and try again.

Console output:

```
Enter your choice (0-7): 2
Enter the Last Name of the customer to remove: a
Customers with Last Name 'a' removed successfully.

Main Menu

1 ) Add Customers
2 ) Remove Customers
3 ) Search Customers
4 ) Update Customer Details
5 ) Show All Customers Details
6 ) Sort Customers (Ascending)
7 ) Sort Customers (Descending)
0 ) Quit

Enter your choice (0-7): 5
No customers to show. Please add a new customer and try again.
```

Data Structures and Algorithm Design (Part 1)

A brief outline of the application

This project aims to design a customer identity book (CIB) application intended to assist banks and financial institutions in establishing their customer identity within the mandatory banking guidelines termed the “Know Your Customer” framework (Elliott et al., 2022). Customer identity is defined by the following mandatory details: the name, date of birth, address, and identification number. In addition, phone numbers and email addresses are collected. The minimum requirements for the data in the labels above are detailed in the next section. The application development involves user interaction with the dataset and it should ensure the implementation of the following operations: create, view, update and delete records. More specifically the CIB application:

- Require the user to enter at least five entries.
- The screen of the application displays functions (create, delete, and etc.) to choose from.
- Create a new customer record by entering the necessary information in the right order.
- Using the customer’s last name (string) input, search for the records.
- Delete records by entering the customer's last name (string).
- Arrange records in a particular sequence and show them on the screen.

Design of data structure and algorithms

The CIB is a database that stores information about a customer's identity as follow:

Table 1. Customer's identity data requirements.

Variable	Description
Id	User ID in the system
Last Name	The customer's Last Name as shown on their ID document. Required field for all customer profiles. * Also known as Surname or Family Name
First Name	The customer's First Name as shown on their ID document. Required field for all customer profiles. * Also known as Given Name.
Date of Birth	Required field for all customer profiles. * The customer's Date of Birth
Identification number	The identity number is printed on all of your national identification documents, such as your ID-card, passport, residents permit etc. A length of the id number might vary. Required field for all customer profiles. ** Also known as a personal code or a national identification number.*
Address	Customer registered address, consisting from street, house number, city/town and country. Required field for all customer profiles.
Phone Number	The valid number starts with 1or8or9 and has the length of 8. Required field for all customer profiles.
Email Address	The format must be username@company.domain format. Username can only contain upper and lowercase letters, numbers, dashes and underscores. Company name can only contain upper and lowercase letters and numbers. Domain can only contain upper and lowercase letters Required field for all customer profiles.

The application provides a system for registering customers in a bank. The system uses dictionary as a data structure, which is a set of key-value pairs, with the condition that each key must be unique in the dictionary (Canning et al., 2023). In this CIB there will be a "Customer" object that helps initialize the Customer data structure, in which every key serves as a specific attribute, where the values for these attributes can be populated with actual customer information when creating instances of this customer construction.

```
“Customer”: {
    "Last_name": <String>,
    "First_name": <String>,
    "Birth_date": <Integer>,
    "Id_number": <Integer>,
    "Address": <String>,
    "E_mail": <String>,
    "Phone_number": <Integer>
}
```

Figure 1. Initialization of data structure

Initialization of an empty list to store customers’ records. And a set the number of customer entries to 5 according to the assignment requirements.

```
customers = []

num = 5
```

Figure 2. Empty list and set of customer entries

Loop to collect customer information from user input

```
FOR each number i in the range up to 'num':

    Create a new customer object named 'customer' as a copy of the
'Customer' data structure

   "Last Name"
   "First Name"
   "Birth Date"
   "Address"
   "ID Number"
   "E-mail"
   "Phone Number"

   Append the customer object to the list

END
```

Figure 3. Customer data collection

Display collected data

```
PRINT header

FOR each 'customer' in 'customers':
    PRINT the values
        id, 'customer["Last_name"]', 'customer["First_name"]',
        'customer["Birth_date"]', 'customer["Address"]',
        'customer["Id_number"]', 'customer["E_mail"]',
        'customer["Phone_number"]'
```

Figure 4. Data displaying

To handle this system, the user will need to perform some manipulation of the customer object. These four operations, called CRUD, consist of creating, reading, updating, and deleting (Chris, 2022). The following pseudocode aims at developing a simple text menu whereby the user can select various options. The program will continue to run until the user decides to stop (chooses the option '0'). The value selected is assigned to the input_choice variable for the use in the remainder of the code not presented in the snippet.

```
PRINT "1 ) Add Customers"
PRINT "2 ) Remove Customers"
PRINT "3 ) Search Customers"
PRINT "4 ) Update Customer Details"
PRINT "5 ) Show All Customers Details"
PRINT "6 ) Sort All Customers Details"
PRINT "0 ) Quit"

'input_choice' to the integer value obtained from user
```

Figure 5. User menu

The create functionality will add a new customer into the database via a trigger, for instance, by choosing “Add Customers" and entering “1” on the screen, calling the relevant method.

```
Continue showing the menu options until the user decides to quit

If 'choice' is equal to '1':

    Create a new customer object named 'customer' as a copy of the
    'Customer' data structure

    Prompt user for input:

      "Last Name"
      "First Name"
      "Birth Date"
      "Address"
      "ID Number"
      "E-mail"
      "Phone Number"

    Append the customer object to the list

Show the menu options
```

Figure 6. Create function

The read feature enables a user to determine whether an entry exists for a given customer in the dataset. This function does not alter any information about the customer, but only enables information about the customer to be fetched.

```
Continue showing the menu options until the user decides to quit

If Choice is equal to '5':

PRINT header

FOR each 'customer' in 'customers':
    PRINT the values
        id, 'customer["Last_name"]', 'customer["First_name"]',
        'customer["Birth_date"]', 'customer["Address"]',
        'customer["Id_number"]', 'customer["E_mail"]',
        'customer["Phone_number"]'

Show the menu options
```

Figure 7. Show function

The update function changes the customer's records using the supplied last-name input. The relevant entries in the dataset table will be transformed after application of the function.

```
Continue showing the menu options until the user decides to quit

    If Choice is equal to '4':

        Prompt user for input: "Last Name"
        Set UpdateLastName to the user's input

        Create a list called UpdateCustomers

        For each customer in customers:
            If customer's Last Name == UpdateLastNamee:
                Add customer to UpdateCustomers

        If UpdateCustomers != empty:
            For each customer in UpdateCustomers:
                Remove customer from customers
                Print customer details
                id, 'customer["Last_name"]', 'customer["First_name"]',
                'customer["Birth_date"]', 'customer["Address"]',
                'customer["Id_number"]', 'customer["E_mail"]',
                'customer["Phone_number"]'

            Print "Success message"
        Else:
            Print "Error message"

Show the menu options
```

Figure 8. Update function

The removal function enables the user to delete customers from a list based on their last names. It asks the user to enter a last name, find customers with that last name, delete them from the original list, and prompt an appropriate feedback message.

```
Continue showing the menu options until the user decides to quit

    If Choice is equal to '2':

        Prompt user for input: "Last Name"
        Set RemoveLastName to the user's input

        Create a list called RemovedCustomers

        For each customer in customers:
            If customer's Last Name == RemoveLastName:
                Add customer to RemovedCustomers

        If RemovedCustomers != empty:
            For each customer in RemovedCustomers:
                Remove customer from customers

            Print "Success message"
        Else:
            Print "Error message"

Show the menu options
```

Figure 9. Delete function

The pseudocode below shows how to search the 'customers' list for customers based on an input from a user (Last Name):

```
Continue showing the menu options until the user decides to quit

    If Choice is equal to '3':

        Prompt user for input: "Last Name"
        Set SearchTerm to the user's input

        For each customer in customers:
            If customer's Last Name == SearchTerm:

                Print values id, 'customer["Last_name"]',
                'customer["First_name"]', 'customer["Birth_date"]',
                'customer["Address"]', 'customer["Id_number"]',
                'customer["E_mail"]', 'customer["Phone_number"]'
            Else:
               Print "Error message"

Show the menu options
```

Figure 10. Search function

The sorting function uses one of the simplest sorting algorithms, the bubble sorting algorithm, on a list of customer records. Customer record sorting is performed based on the "Last_name" attribute in each dictionary. Consequently, the algorithm sequentially checks each set of adjacent elements in a list, and if the elements are swapped incorrectly, then the algorithm performs a swap (Kumar, 2022). The algorithm stops when further swaps are no longer required (Lopez, 2022).

```
function bubble_sort(customers):
    n = length of customers
    # Outer loop for multiple passes
    for i from 0 to n - 1:
        # Inner loop for each pass
        for j from 0 to n - i - 1:
            # Compare Last_name of the current customer to the next
            customer and swap if necessary
            if customers[j]["Last_name"] > customers[j + 1]["Last_name"]:
                swap customers[j] with customers[j + 1]

Continue showing the menu options until the user decides to quit

If Choice is equal to '6':

    bubble_sort(customers)

    # Print each customer's details on a separate line
    PRINT header
    FOR each 'customer' in 'customers':
        PRINT the values
            id, 'customer["Last_name"]', 'customer["First_name"]',
            'customer["Birth_date"]', 'customer["Address"]',
            'customer["Id_number"]', 'customer["E_mail"]',
            'customer["Phone_number"]'

Show the menu options
```

Figure 11. Sorting function

Test plan

The test plan includes detailed test scenarios that prove the proper operation and validity of the program being implemented in the customer identity book.

Table 2. CIB application test scenario

Scenario	Test case	Expected Outcome
Customer Management Operations		
Add 5 customer entries according to the requirements	Customer = { "Last_name": "Doe", "First_name": "Thomas", "Birth_date": 19900101, "Address": "Oak 1, Vilnius, LT", "Id_number": 111111, "E_mail": "thomas.doe@email.com", "Phone_number": 511511511 "Last_name": "Nuu", "First_name": "Mark", "Birth_date": 19900202, "Address": "Oak 2, Vilnius, LT", "Id_number": 222222, "E_mail": "thomas.doe@email.com", "Phone_number": 512512512 "Last_name": "Wue", "First_name": "David", "Birth_date": 19900303, "Address": "Oak 3, Vilnius, LT", "Id_number": 333333, "E_mail": "thomas.doe@email.com", "Phone_number": 513513513 "Last_name": "Liu", "First_name": "Joe", "Birth_date": 19900404, "Address": "Oak 4, Vilnius, LT", "Id_number": 444444, "E_mail": "thomas.doe@email.com", "Phone_number": 514514514	The program should successfully add customers to the list from the beginning. ID Name Birth date Address 1 Doe Thomas 19900101 Oak 1, Vilnius, LT 2 Nuu Mark 19900202 Oak 2, Vilnius, LT 3 Wue David 19900303 Oak 3, Vilnius, LT 4 Liu Joe 19900404 Oak 4, Vilnius, LT 5 Key Sven 19900505 Oak 5, Vilnius, LT ID Number E-mail Phone Number 111111 thomas.doe@email.com 511511511 222222 mark.nuu@email.com 512512512 333333 david.wue@email.com 513513513 444444 joe.liu@email.com 514514514 555555 sven.key@email.com 515515515

	"Last_name": "Key", "First_name": "Sven", "Birth_date": 19900505, "Address": "Oak 5, Vilnius, LT", "Id_number": 555555, "E_mail": "thomas.doe@email.com", "Phone_number": 515515515 }	
Add Customers **Objective**: Verify that customers can be successfully added.	1. Choose option '1' to add customers. 2. Enter information for multiple customers: Customer = { "Last_name": "Boo", "First_name": "Drake", "Birth_date": 19900404, "Address": "Oak 4, Vilnius, LT", "Id_number": 444444, "E_mail": "thomas.doe@email.com", "Phone_number": 514514514 } 3. Verify that the customers are added to the list.	The program should successfully add customers to the list. ID Name Birth date Address 6 Boo Drake 19900606 Oak 6, Vilnius, LT ID Number E-mail Phone Number 666666 drake.boo@email.com 516516516
Remove Customers **Objective**: Verify that customers can be successfully removed.	1. Choose option '2' to remove customers. 2. Enter the last name of an existing customer. "Last_name": "Liu" 3. Verify that the customer is removed from the list.	The program should successfully remove the specified customer.
Search Customers **Objective**: Verify that the program	1. Choose option '3' to search for customers. 2. Enter a search term (last name). "Last_name": "Nuu"	The program should successfully display customer information based on the search term. ID Name Birth date Address

can successfully search for customers	3. Verify that the program displays the relevant customer information.	2 Nuu Mark 19900202 Oak 2, Vilnius, LT ID Number E-mail Phone Number 222222 mark.nuu@email.com 512512512
Update Customer Details **Objective:** Verify that customer details can be successfully updated.	1. Choose option '4' to update customer details. 2. Enter the last name of an existing customer. "Last_name": "Doe" 3. Verify that the program displays the customer details for confirmation. 4. Enter updated information. 5. Verify that the customer details are updated.	The program should successfully update the specified customer's details.
Show All Customers Details **Objective:** Verify that the program can display all customer details.	1. Choose option '5' to show all customer details. 2. Verify that the program displays a formatted list of all customers.	The program should successfully display all customer details. ID Name Birth date Address 1 Doe Thomas 19900101 Oak 1, Vilnius, LT 2 Wue David 19900303 Oak 3, Vilnius, LT 3 Liu Joe 19900404 Oak 4, Vilnius, LT 4 Key Sven 19900505 Oak 5, Vilnius, LT 5 Boo Drake 19900606 Oak 6, Vilnius, LT ID Number E-mail Phone Number 111111 thomas.doe@email.com 511511511 333333 david.wue@email.com 513513513 444444 joe.liu@email.com 514514514 555555 sven.key@email.com 515515515 666666 drake.boo@email.com 516516516
Sort All Customers Details	1. Choose option '6' to sort all customer details. 2. Verify that the program prints a success message.	The program should successfully sort and display all customer details based on last name.

Objective: Verify that the program can sort all customer details based on last name.	3. Verify that the customer details are displayed in sorted order.	ID Name Birth date Address 1 Boo Drake 19900606 Oak 6, Vilnius, LT 2 Doe Thomas 19900101 Oak 1, Vilnius, LT 3 Key Sven 19900505 Oak 5, Vilnius, LT 4 Liu Joe 19900404 Oak 4, Vilnius, LT 5 Wue David 19900303 Oak 3, Vilnius, LT ID Number E-mail Phone Number 666666 drake.boo@email.com 516516516 111111 thomas.doe@email.com 511511511 555555 sven.key@email.com 515515515 444444 joe.liu@email.com 514514514 333333 david.wue@email.com 513513513
User Input Validation:		
Menu Choices **Objective:** Ensure that the program handles invalid menu choices gracefully.	1. Enter an invalid menu choice (e.g., '7'). 2. Verify that the program prints an error message. 3. Repeat the test with other invalid choices.	The program should inform the user of the invalid choice.
Numeric Input for Birth Date and ID Number **Objective:** Verify that the program handles numeric input for birth date and ID number.	1. Enter non-numeric characters for birth date or ID number. 2. Verify that the program prompts the user for valid numeric input. 3. Enter valid numeric input for birth date and ID number.	The program should handle non-numeric input gracefully and request valid numeric input.

Numeric Input for Phone Number **Objective:** Verify that the program handles numeric input for phone number.	1. Enter non-numeric characters for phone number. 2. Verify that the program prompts the user for valid numeric input. 3. Enter valid numeric input for phone number.	The program should handle non-numeric input gracefully and request valid numeric input.
Valid Input for Email **Objective:** Verify that the program handles valid input for email.	1. Enter email without “@” symbol. 2. Verify that the program prompts the user for valid email input. 3. Enter valid format input for email.	The program should handle non-valid input gracefully and request valid email input

Conclusion

The customer identity book is a simple customer management application, having a menu-driven console interface, supported by user-friendly prompts. Users can add, remove, update, search and sort customer records within the program. The sorting function uses one of the simplest bubble sorting algorithms, on customer records, based on the last name attribute. Users can view, update, insert and delete customers' records. The application is designed to process user inputs and ensure feedback and comprehensive outputs. Besides the user interface, the program code is effectively structured with the help of functions and loops. As a result, the project of CIB fulfils its intended purpose of managing and interacting with customers' records.

Reference list

Canning, J., Broder, A. and Lafore, R. (2023) *Data Structures Et algorithms in Python*. Boston: Addison-Wesley.

Chris, K. (2022) *CRUD operations – what is crud?*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/crud-operations-explained/ [Accessed: 08 January 2024].

Elliott, K., Coopamootoo, K., Curran, E., Ezhilchelvan, P., Finnigan, S., Horsfall, D., Ma, Z., Ng, M., Spiliotopoulos, T., Wu, H., and van Moorsel, A. (2022) 'Know your customer: Balancing innovation and regulation for financial inclusion', *Data & Policy*, 4. doi:10.1017/dap.2022.23.

Kumar, A.D. (2022) *Bubble sort in python with complexity analysis*, *LinkedIn*. Available at: https://www.linkedin.com/pulse/bubble-sort-python-complexity-analysis-amara-dinesh-kumar/ [Accessed: 17 January 2024].

Lopez, E. (2022) *Bubble sort - how it works, Psuedocode and C++ & Python implementation*, *Medium*. Available at: https://medium.com/codex/bubble-sort-how-it-works-psuedocode-and-c-python-implementation-c45306d44827 [Accessed: 16 January 2024].

Conclusion

The customer identity book is a simple customer management application, having a menu-driven console interface, supported by user-friendly prompts. Users can add, remove, update, search and sort customer records within the program. The sorting function uses one of the simplest bubble sorting algorithms, on customer records, based on the last name attribute. Users can view, update, insert and delete customers' records. The application is designed to process user inputs and ensure feedback and comprehensive outputs. Besides the user interface, the program code is effectively structured with the help of functions and loops. As a result, the project of CIB fulfils its intended purpose of managing and interacting with customers' records.