Assignment 1 Part 1 of 3
Launching into Computer Science

*Word count: 785*

# Data Structures and Algorithm Design

## 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.<br>Required field for all customer profiles.<br>* Also known as Surname or Family Name |
| First Name | The customer's First Name as shown on their ID document.<br>Required field for all customer profiles.<br>* Also known as Given Name. |
| Date of Birth | Required field for all customer profiles.<br>* 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.<br>A length of the id number might vary.<br>Required field for all customer profiles.<br>** Also known as a personal code or a national identification number.* |
| Address | Customer registered address, consisting from street, house number, city/town and country.<br>Required field for all customer profiles. |
| Phone Number | The valid number starts with 1or8or9 and has the length of 8.<br>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<br>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 = {<br>"Last_name": "Doe",<br>"First_name": "Thomas",<br>"Birth_date": 19900101,<br>"Address": "Oak 1, Vilnius, LT",<br>"Id_number": 111111,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 511511511<br><br>"Last_name": "Nuu",<br>"First_name": "Mark",<br>"Birth_date": 19900202,<br>"Address": "Oak 2, Vilnius, LT",<br>"Id_number": 222222,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 512512512<br><br>"Last_name": "Wue",<br>"First_name": "David",<br>"Birth_date": 19900303,<br>"Address": "Oak 3, Vilnius, LT",<br>"Id_number": 333333,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 513513513<br><br>"Last_name": "Liu",<br>"First_name": "Joe",<br>"Birth_date": 19900404,<br>"Address": "Oak 4, Vilnius, LT",<br>"Id_number": 444444,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 514514514 | The program should successfully add customers to the list from the beginning.<br><br>ID Name Birth date Address<br>1 Doe Thomas 19900101 Oak 1, Vilnius, LT<br>2 Nuu Mark 19900202 Oak 2, Vilnius, LT<br>3 Wue David 19900303 Oak 3, Vilnius, LT<br>4 Liu Joe 19900404 Oak 4, Vilnius, LT<br>5 Key Sven 19900505 Oak 5, Vilnius, LT<br><br>ID Number E-mail Phone Number<br>111111 thomas.doe@email.com 511511511<br>222222 mark.nuu@email.com 512512512<br>333333 david.wue@email.com 513513513<br>444444 joe.liu@email.com 514514514<br>555555 sven.key@email.com 515515515 |

| | | |
|---|---|---|
| | "Last_name": "Key",<br>"First_name": "Sven",<br>"Birth_date": 19900505,<br>"Address": "Oak 5, Vilnius, LT",<br>"Id_number": 555555,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 515515515<br>} | |
| **Add Customers**<br><br>**Objective:** Verify that customers can be successfully added. | 1. Choose option '1' to add customers.<br>2. Enter information for multiple customers:<br>Customer = {<br>"Last_name": "Boo",<br>"First_name": "Drake",<br>"Birth_date": 19900404,<br>"Address": "Oak 4, Vilnius, LT",<br>"Id_number": 444444,<br>"E_mail": "thomas.doe@email.com",<br>"Phone_number": 514514514<br>}<br>3. Verify that the customers are added to the list. | The program should successfully add customers to the list.<br><br>ID Name Birth date Address<br>6 Boo Drake 19900606 Oak 6, Vilnius, LT<br><br>ID Number E-mail Phone Number<br>666666 drake.boo@email.com 516516516 |
| **Remove Customers**<br><br>**Objective:** Verify that customers can be successfully removed. | 1. Choose option '2' to remove customers.<br>2. Enter the last name of an existing customer.<br>"Last_name": "Liu"<br>3. Verify that the customer is removed from the list. | The program should successfully remove the specified customer. |
| **Search Customers**<br><br>**Objective:** Verify that the program | 1. Choose option '3' to search for customers.<br>2. Enter a search term (last name).<br>"Last_name": "Nuu" | The program should successfully display customer information based on the search term.<br><br>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<br><br>ID Number E-mail Phone Number<br>222222 mark.nuu@email.com 512512512 |
| **Update Customer Details**<br><br>**Objective:** Verify that customer details can be successfully updated. | 1. Choose option '4' to update customer details.<br>2. Enter the last name of an existing customer. "Last_name": "Doe"<br>3. Verify that the program displays the customer details for confirmation.<br>4. Enter updated information.<br>5. Verify that the customer details are updated. | The program should successfully update the specified customer's details. |
| **Show All Customers Details**<br><br>**Objective:** Verify that the program can display all customer details. | 1. Choose option '5' to show all customer details.<br>2. Verify that the program displays a formatted list of all customers. | The program should successfully display all customer details.<br><br>ID Name Birth date Address<br>1 Doe Thomas 19900101 Oak 1, Vilnius, LT<br>2 Wue David 19900303 Oak 3, Vilnius, LT<br>3 Liu Joe 19900404 Oak 4, Vilnius, LT<br>4 Key Sven 19900505 Oak 5, Vilnius, LT<br>5 Boo Drake 19900606 Oak 6, Vilnius, LT<br><br>ID Number E-mail Phone Number<br>111111 thomas.doe@email.com 511511511<br>333333 david.wue@email.com 513513513<br>444444 joe.liu@email.com 514514514<br>555555 sven.key@email.com 515515515<br>666666 drake.boo@email.com 516516516 |
| **Sort All Customers Details** | 1. Choose option '6' to sort all customer details.<br>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<br>1 Boo Drake 19900606 Oak 6, Vilnius, LT<br>2 Doe Thomas 19900101 Oak 1, Vilnius, LT<br>3 Key Sven 19900505 Oak 5, Vilnius, LT<br>4 Liu Joe 19900404 Oak 4, Vilnius, LT<br>5 Wue David 19900303 Oak 3, Vilnius, LT<br><br>ID Number E-mail Phone Number<br>666666 drake.boo@email.com 516516516<br>111111 thomas.doe@email.com 511511511<br>555555 sven.key@email.com 515515515<br>444444 joe.liu@email.com 514514514<br>333333 david.wue@email.com 513513513 |
| **User Input Validation:** | | |
| **Menu Choices**<br><br>**Objective:** Ensure that the program handles invalid menu choices gracefully. | 1. Enter an invalid menu choice (e.g., '7').<br>2. Verify that the program prints an error message.<br>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**<br><br>**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.<br>2. Verify that the program prompts the user for valid numeric input.<br>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**<br><br>**Objective:** Verify that the program handles numeric input for phone number. | 1. Enter non-numeric characters for phone number.<br>2. Verify that the program prompts the user for valid numeric input.<br>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**<br><br>**Objective:** Verify that the program handles valid input for email. | 1. Enter email without “@” symbol.<br>2. Verify that the program prompts the user for valid email input.<br>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].