

**Technical University of Sofia**

**SAA Course Project**

**Alexander Alexandrov Svilarov**

***Faculty Number*** *– 371223006*

*Group 90a*

***Smart Systems and Artificial Intelligence***

*1.12.2024*

**1. Explanatory text of the program**

**Explanation of the Bank Management System Program**

This project implements a **Bank Management System** where users can manage customer data, including adding new customers, displaying customer details, sorting customer records, and filtering them based on account balance.

**Program Structure**

1. **Header Inclusions:**

* The program includes standard headers like <iostream>, <vector>, <random>, and <sstream>.
* A custom **Customer.h** file is also included, defining the **Customer** class with attributes: ***ID****,* ***name****,* ***accountNumber****,* ***accountAmount*** *and* ***SSN****.*

1. **User Interface:**
   * The main interface is a menu displayed in the console, allowing users to select one of five options:
     1. **Add a Customer**
     2. **Display All Customers**
     3. **Sort Customers by Account Amount (Descending)**
     4. **Sort Customers by Name (Ascending)**
     5. **Find all customers with more than a given amount of money**

The user’s input is processed in a loop until the user selects **option 0** to exit.

**Key Functionalities**

1. **Adding a Customer:**
   * A new customer is created by generating a unique **ID** using the **generateID()** function, which generates random hexadecimal characters.
   * The user inputs the **name**, **account balance**, and **SSN**, and the data is added to the

**customers** vector.

1. **Displaying Customers:**

* The program iterates through the **customers** vector and calls the **printInfo()** method of each Customer object to display their details.

1. **Sorting Customers:**

* Two sorting methods are implemented:
  + - **By Account Amount (Descending)**: Bubble sort logic (although misplaced in option 3, since it sorts by name).
    - **By Name (Ascending)**: The **quicksort()** function recursively partitions the vector based on the names of the customers, ensuring efficient sorting.

1. **Filtering Customers:**

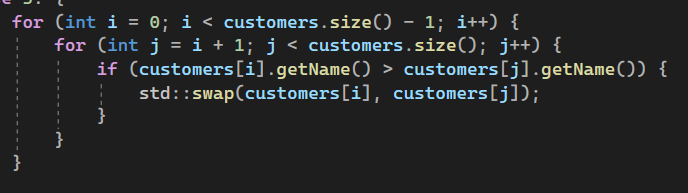
* The program filters and displays customers whose account balance exceeds a user-defined limit using a simple loop and conditional check.

**Supporting Functions**

* **generateID():** Creates a 32-character hexadecimal ID formatted like a GUID.
* **partition() and quicksort():** Implement a standard quicksort algorithm for sorting the customer list by name.

1. **Explanatary text for the sorting algorithms.**

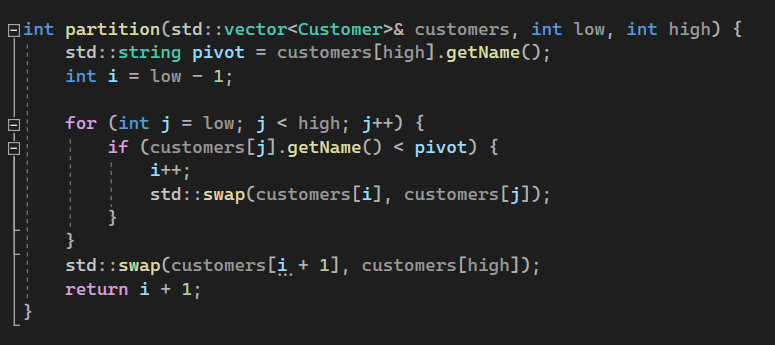
***2.1 Bubble sort***

****

. Overview of the algorithm

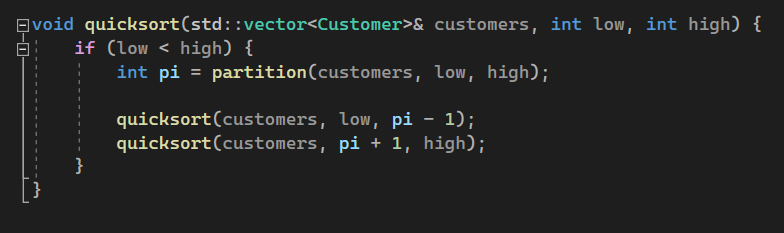
* **Outer Loop (i loop)**:
  + **for (int i = 0; i < customers.size() - 1; i++):**The outer loop runs from the first customer (i = 0) to the second-to-last customer (customers.size() - 1). It controls how many passes are made through the list. Each pass brings the smallest (alphabetically first) name closer to the start.
* **Inner Loop (j loop)**:
  + **for (int j = i + 1; j < customers.size(); j++):**  
    The inner loop starts from i + 1 and checks each subsequent customer. This loop compares the current customer at position i with the customers that come after it.
* **Comparison Condition**:
  + **if (customers[i].getName() > customers[j].getName()):**This condition checks if the name of the customer at position i comes **later** alphabetically than the name at position j. If it does, a swap is needed.
* **Swapping Elements**:
  + **std::swap(customers[i], customers[j]):**If the condition is true, the std::swap() function exchanges the positions of these two customers, placing the alphabetically smaller name first.

***2.2 Quicksort***

****

. Overview of the partition method

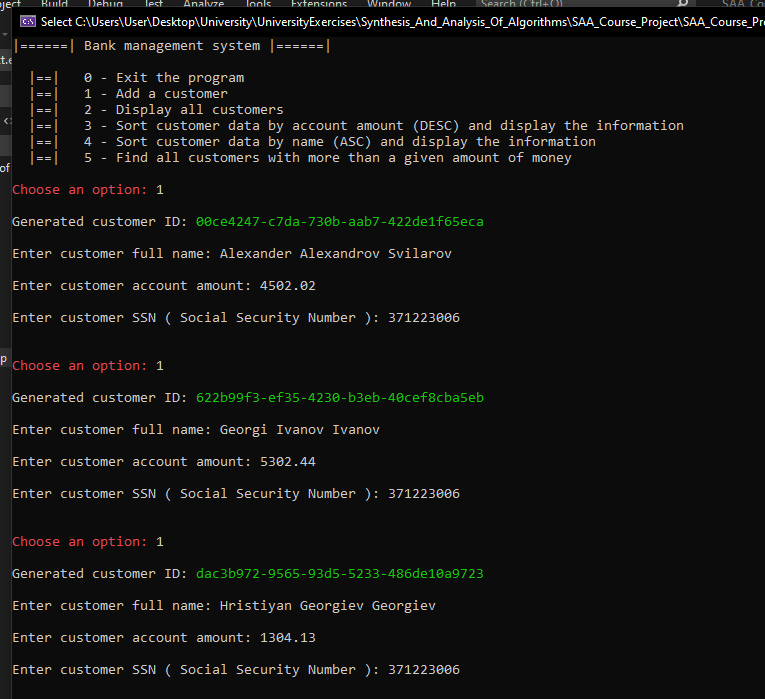
* **Pivot Selection**: The pivot point is chosen as the last element of the sublist (**customers[high].getName()**).
* **Element Comparison**: The loop compares each customer's name to the pivot. If a name is smaller than the pivot, it's swapped to the left.
* **Final Pivot Placement**: The pivot is placed in its correct sorted position, and its index is returned.

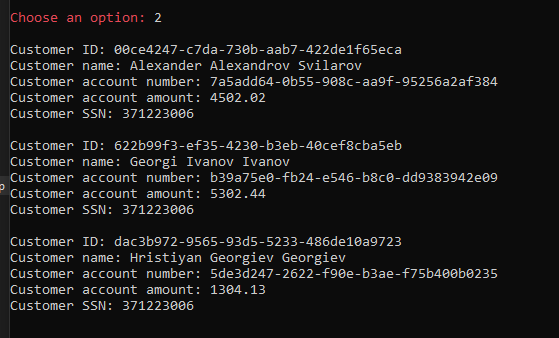
****

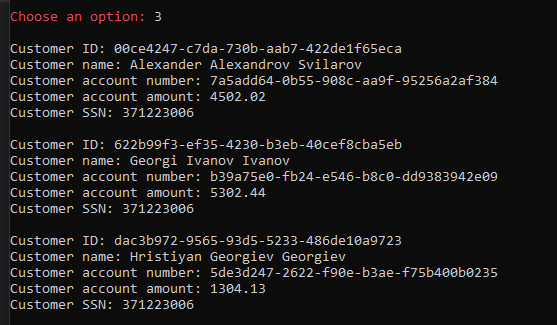
1. Overview of the main quicksort method

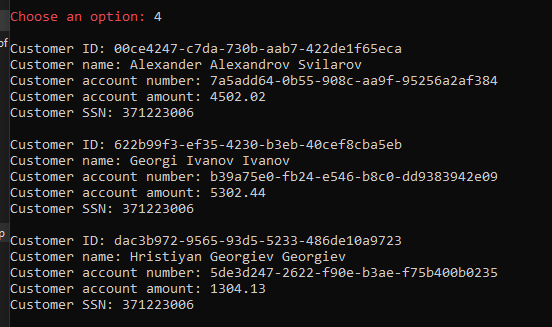
* **Recursive division**: The list is divided into sublists before and after the pivot. The process continues until each sublist has one or no elements.
* **Base Case**: Sorting stops when **low** is not less than **high**.

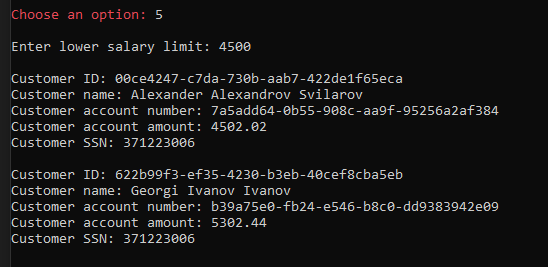
**3. Sample input and output**

****

****

****

****

****