Table of Contents

1.0 Introduction 3
1.1 Assumption 3
1.1.1 Tenant 3
1.1.2 Manager 3
1.1.3 Admin 3
1.2 Data Structure Algorithm 4
1.2.1 Doubly linked list 4
1.2.2 Queue 10
1.3 Class 13
1.3.1 Admin class 13
1.3.2 Manager class 14
1.3.3 Property 15
1.3.4 Tenant 17
1.4 Initialization of Data Structures 18
1.4.1 initializeAdmin 18
1.4.2 initializeManager 18
1.4.3 initializeTenant 19
1.4.4 initializeFavouriteListAll 19
1.4.5 initializeFavouriteListTenant 20
1.4.6 initializeRentRequest 21
1.4.7 initializeRentingHistory 21
1.4.8 initializeProperty 22
1.5 Utils 24
1.5.1 toLowerCase/toUpperCase 24
1.5.2 askUserUsername/askUserPassword 25
1.5.3 searchByUsername 25
1.5.4 exitFunction 26
2.0 Implementation and Result 27
2.1 Source code for each algorithm 27
2.1.1 Bubble Sort 27
2.1.2 Quick Sort 30
2.1.3 Linear Search 33
2.1.4 Binary Search 35
2.1.5 Comparison Sort-time 37

2.1.6 Comparison Search Time ... 38
3.0 System Input /Output ... 40
3.1 All user ... 40
3.1.1 Main Menu ... 40
3.1.2 Register as tenant ... 40
3.1.3 Login ... 41
3.1.4 Display property ... 41
3.1.5 Exit function ... 42
3.2 Admin ... 43
3.2 1 Add new manager ... 43
3.2.2 Modify manager status ... 44
3.2.3 View tenant detail ... 44
3.2.4 View property information ... 45
3.3 Managers ... 47
3.3.1 Login ... 47
3.3.2 Display all Registered Tenant’s Details ... 47
3.3.3 Search Tenant’s Details ... 48
3.3.4 Delete Tenant Accounts based on Inactivity Status. ... 48
3.3.5 Summarize the Top 10 properties from the favourite properties list. ... 49
3.3.6 Manage Tenancy Process ... 50
3.3.7 Manage Payment of the confirmed tenancy. ... 52
3.3.8 Logout ... 54
3.4 Tenants ... 55
3.4.1 Tenant main menu ... 55
3.4.2 Sort Property ... 55
3.4.2 Add favourite property ... 57
3.4.3 Search Property ... 57
3.4.4 Rent Request from Favourite Property ... 59
3.4.5 Display Rent History List ... 59
4.0 Conclusion ... 61
4.1 Summary with Limitation ... 61
4.2 Future Work ... 62
4.3 Reflection ... 62
5.0 References ... 63

1.0 Introduction

Asia Pacific Home (APH) is working on a new project to make renting homes in the Klang Valley area of Malaysia easier and more efficient. By teaming up with APU tech-solution, they're creating a system that will take property listings from websites like Advert web (mudah) and organize them in one place. The new system will allow tenants to sort through over 10,000 properties, save their favourite ones, and even request to rent them. Managers will be able to see tenant details, handle rent requests, and payments, while administrators can manage the user accounts. This project is all about using technology to make finding and renting a home as simple as possible.

1.1 Assumption

- All property details are completed.
- One property could be saved as a favorite by multiple tenants.
- When displaying a property, tenancy Progress is set as '---' in default.
- There is ready 55 properties in the main doubly linked list favorite list.
- There are two ready property rent requests in the queue.
- Renting history and managed payment will be stored in the same queue list.

1.1.1 Tenant

- All tenants could remember their usernames and password.
- All tenants fill in their details correctly.
- Tenants will be paid when placing a rent.
- There are three active tenant accounts in the doubly linked list.

1.1.2 Manager

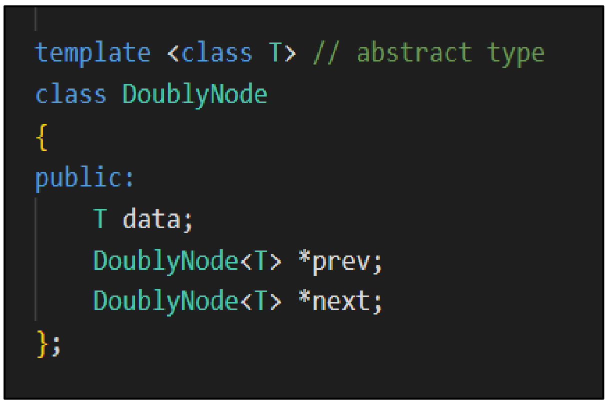
- All managers could remember their usernames and password.
- Tenancy progress can only be managed when a tenant places a rent request.
- The payment function can only be executed when tenancy progress has been updated to approved or rejected.
- Payment function is executed as accept the tenant's payment or reject. The system changes the tenancy progress to active when the payment is accepted.
- There is four ready manager accounts in the doubly linked list.

1.1.3 Admin

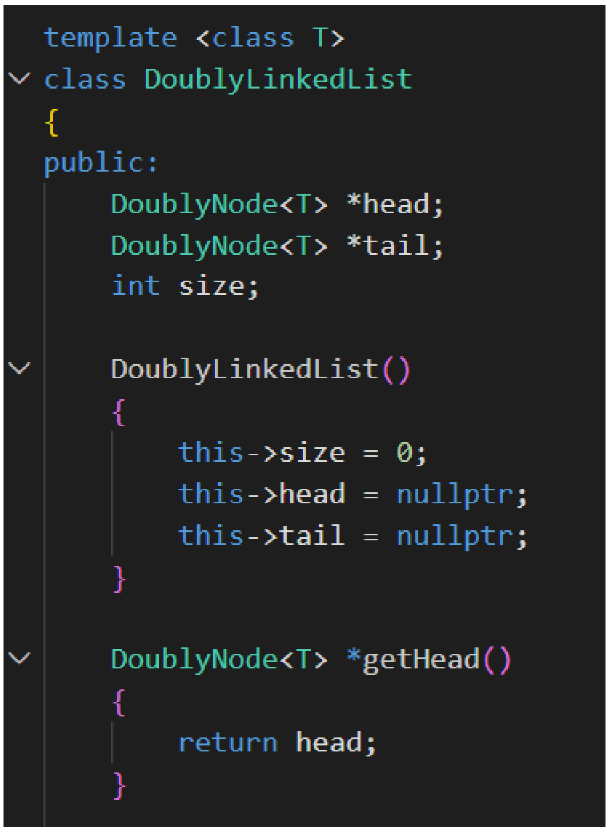
- All admins could remember their usernames and password.
- There is a ready admin account with these details: (1, "admin," "admin123", "male")

1.2 Data Structure Algorithm

1.2.1 Doubly linked list



```cpp
template <class T> // abstract type
class DoublyNode
{
public:
    T data;
    DoublyNode<T> *prev;
    DoublyNode<T> *next;
};
```

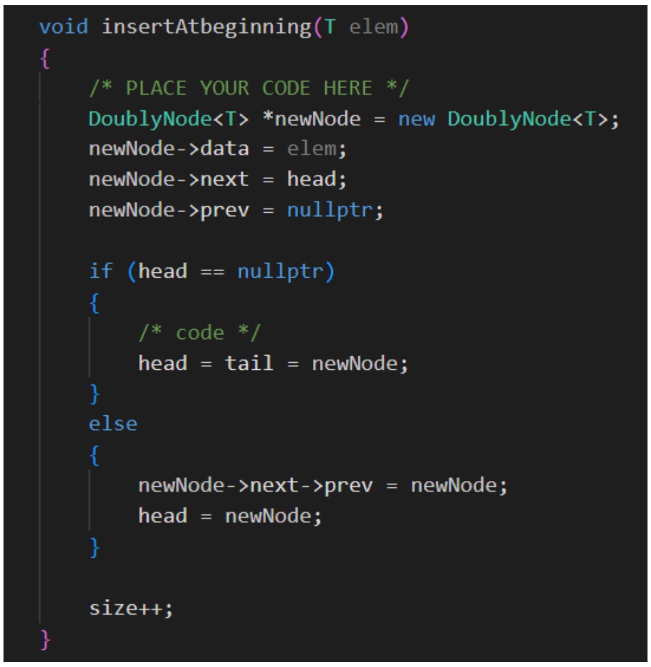


```cpp
template <class T>
class DoublyLinkedList
{
public:
    DoublyNode<T> *head;
    DoublyNode<T> *tail;
    int size;

    DoublyLinkedList()
    {
        this->size = 0;
        this->head = nullptr;
        this->tail = nullptr;
    }

    DoublyNode<T> *getHead()
    {
        return head;
    }
```

Figure 1:Doubly Linked List class

A Doubly Linked List, represented in the given code, is a complex structure where each element contains data and pointers to both the next and previous elements in the sequence, allowing navigation in both directions. The code defines two template classes: Doubly Node, representing an individual node with data of an abstract type T and pointers Prev and next, and Doubly Linked List, representing the list itself with a head pointer to the first element, a tail pointer to the last element, and an integer size for tracking the number of elements. The constructor initializes these elements, and the getHead() function provides access to the list's elements. Together, these classes create a flexible structure that enables efficient insertions, deletions, and traversals.

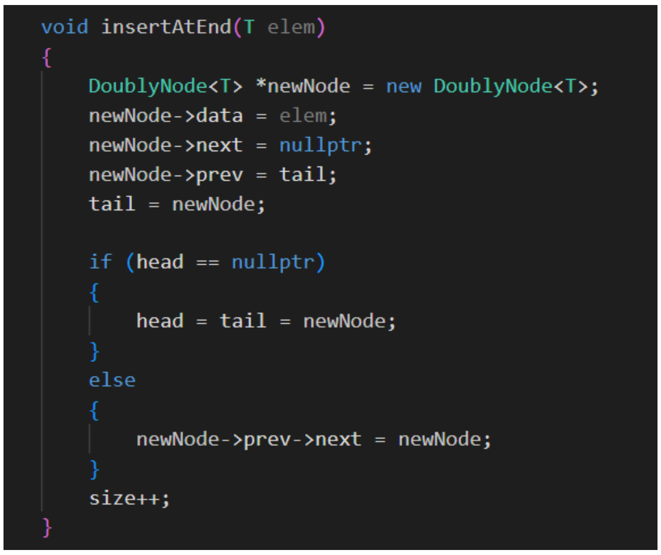
Function inside doubly linked list



```
void insertAtbeginning(T elem)
{
    /* PLACE YOUR CODE HERE */
    DoublyNode<T> *newNode = new DoublyNode<T>;
    newNode->data = elem;
    newNode->next = head;
    newNode->prev = nullptr;

    if (head == nullptr)
    {
        /* code */
        head = tail = newNode;
    }
    else
    {
        newNode->next->prev = newNode;
        head = newNode;
    }

    size++;
}
```

Figure 2:Insert at beginning function

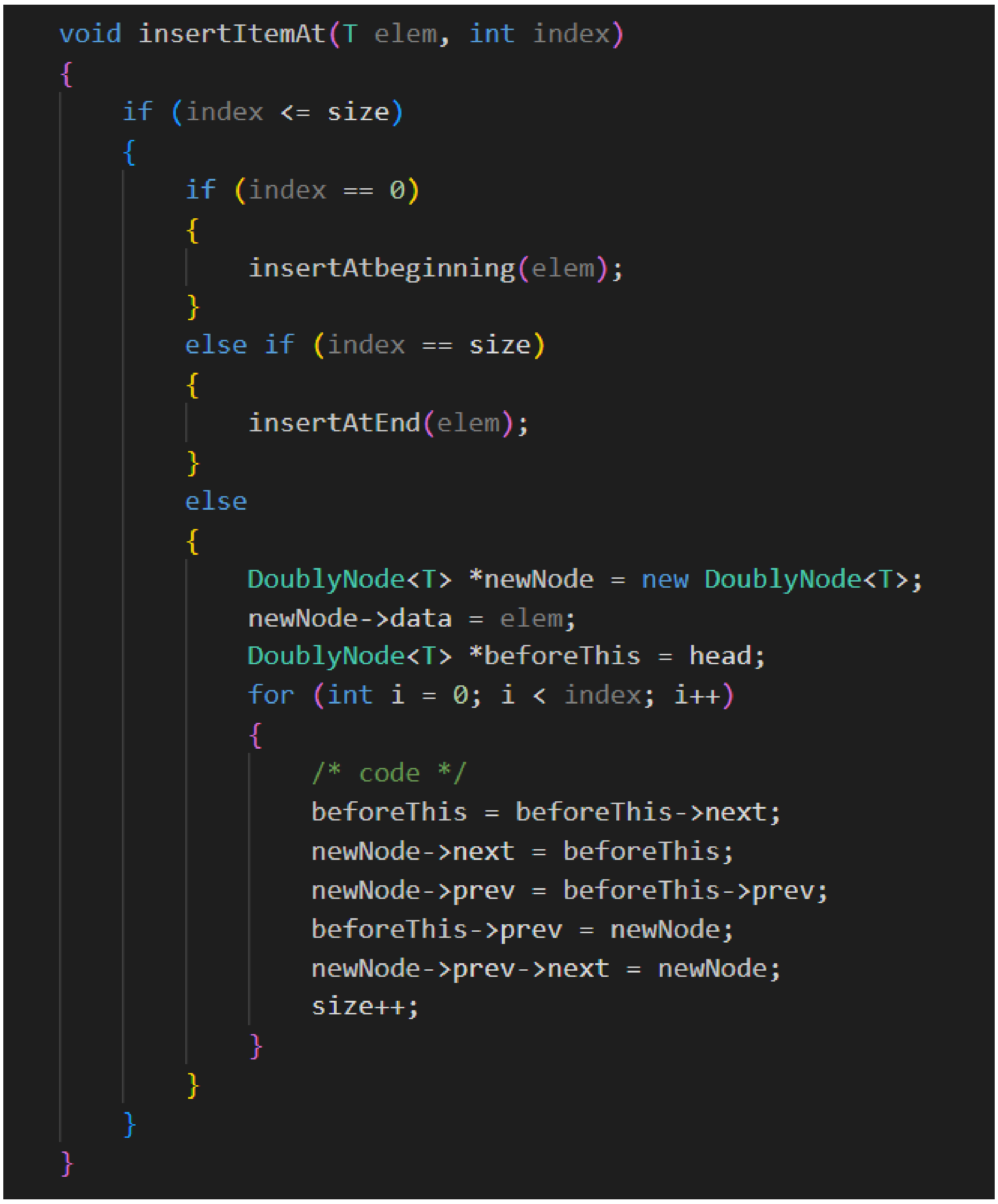
The code above show the insert function in the doubly linked list class to allow user to insert the element into the doubly linked list at the beginning . The insertAtbeginning function adds an element at the start of the doubly linked list. It creates a new node, sets its data, and adjusts the previous and next pointers to link it properly. If the list is empty, the new node becomes both the head and tail. The size is then incremented.



```
void insertAtEnd(T elem)
{
    DoublyNode<T> *newNode = new DoublyNode<T>;
    newNode->data = elem;
    newNode->next = nullptr;
    newNode->prev = tail;
    tail = newNode;

    if (head == nullptr)
    {
        head = tail = newNode;
    }
    else
    {
        newNode->prev->next = newNode;
    }
    size++;
}
```

Figure 3:Insert at the end function

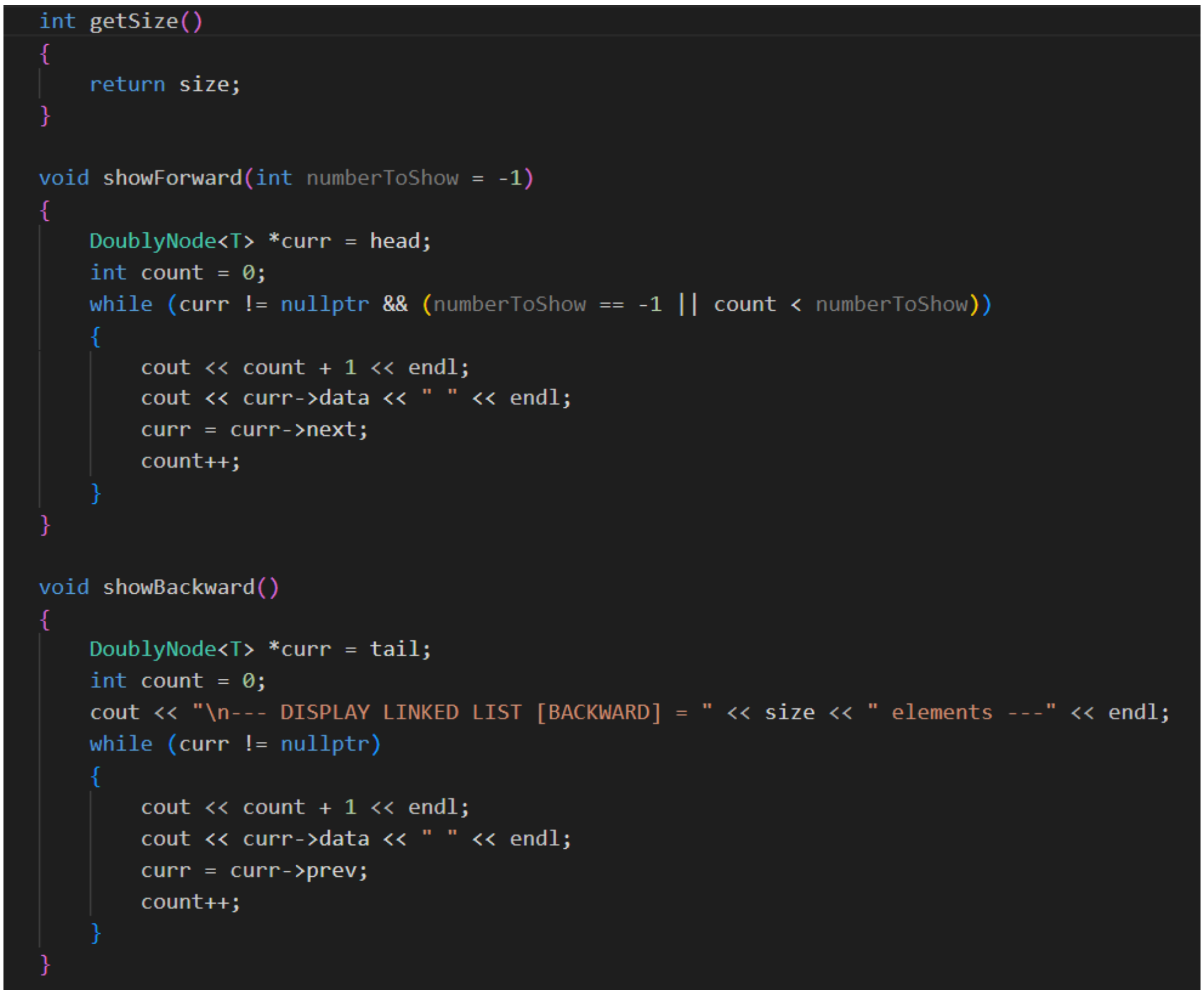
The insertAtEnd function appends an element to the end of the doubly linked list. It creates a new node and sets its data, linking it as the new tail. If the list is empty, the new node becomes both the head and tail. Otherwise, it properly adjusts the next and previous pointers. The size is then incremented.



```
void insertItemAt(T elem, int index)
{
    if (index <= size)
    {
        if (index == 0)
        {
            insertAtbeginning(elem);
        }
        else if (index == size)
        {
            insertAtEnd(elem);
        }
        else
        {
            DoublyNode<T> *newNode = new DoublyNode<T>;
            newNode->data = elem;
            DoublyNode<T> *beforeThis = head;
            for (int i = 0; i < index; i++)
            {
                /* code */
                beforeThis = beforeThis->next;
                newNode->next = beforeThis;
                newNode->prev = beforeThis->prev;
                beforeThis->prev = newNode;
                newNode->prev->next = newNode;
                size++;
            }
        }
    }
}
```

Figure 4:Insert item at function

The insertItemAt function inserts an element at a specific index in the doubly linked list. If the index is 0, it inserts at the beginning, and if the index is equal to the size, it inserts at the end. For other indices, it traverses to the correct position and adjusts the next and previous pointers of the neighbouring nodes to insert the new element. The size is then incremented.



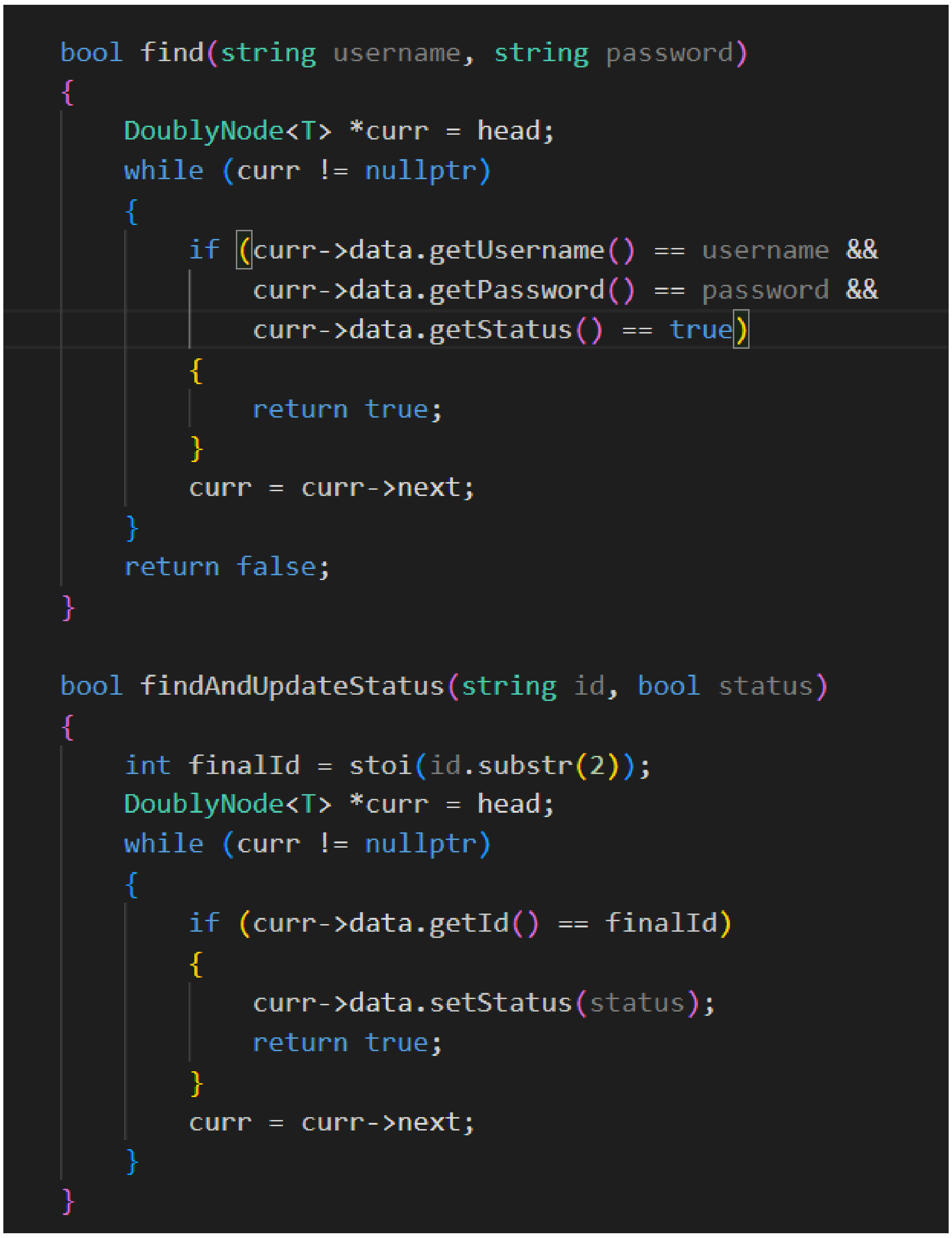
```
int getSize()
{
    return size;
}

void showForward(int numberToShow = -1)
{
    DoublyNode<T> *curr = head;
    int count = 0;
    while (curr != nullptr && (numberToShow == -1 || count < numberToShow))
    {
        cout << count + 1 << endl;
        cout << curr->data << " " << endl;
        curr = curr->next;
        count++;
    }
}

void showBackward()
{
    DoublyNode<T> *curr = tail;
    int count = 0;
    cout << "\n--- DISPLAY LINKED LIST [BACKWARD] = " << size << " elements ---" << endl;
    while (curr != nullptr)
    {
        cout << count + 1 << endl;
        cout << curr->data << " " << endl;
        curr = curr->prev;
        count++;
    }
}
```

Figure 5: getSize(), showForward() and showBackward() function

getSize(): This function will then return the size of this doubly linked list.

showForward() :This function prints the elements of the doubly linked list in a forward direction. If an integer numberToShow is provided, it will print that many elements from the beginning of the list. If the value is -1 (default), it prints all elements.

showBackward(): This function prints the elements of the doubly linked list in a reverse or backward direction, starting from the tail and moving towards the head, displaying all the elements of the list.

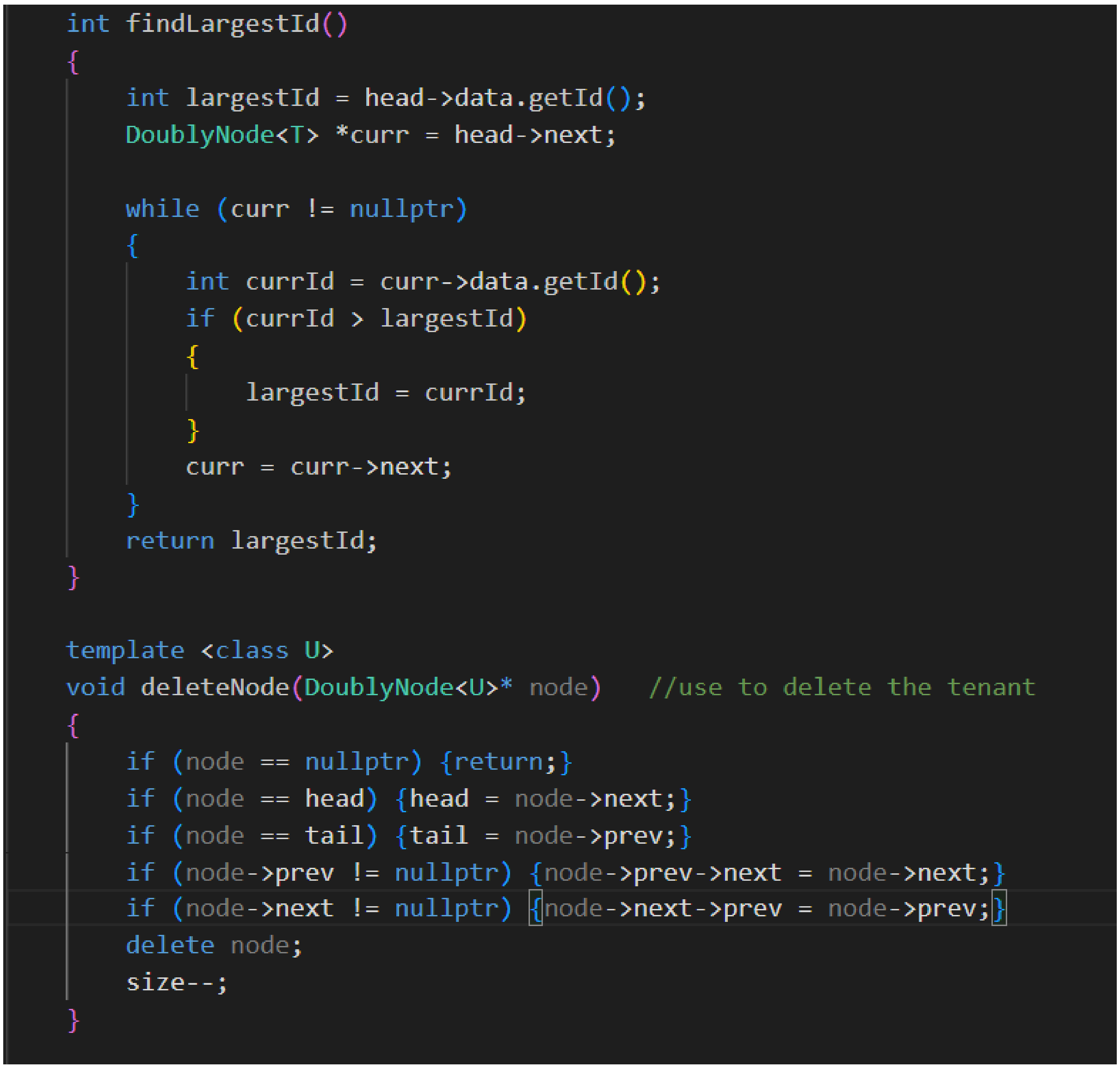


```
bool find(string username, string password)
{
    DoublyNode<T> *curr = head;
    while (curr != nullptr)
    {
        if (curr->data.getUsername() == username &&
            curr->data.getPassword() == password &&
            curr->data.getStatus() == true)
        {
            return true;
        }
        curr = curr->next;
    }
    return false;
}

bool findAndUpdateStatus(string id, bool status)
{
    int finalId = stoi(id.substr(2));
    DoublyNode<T> *curr = head;
    while (curr != nullptr)
    {
        if (curr->data.getId() == finalId)
        {
            curr->data.setStatus(status);
            return true;
        }
        curr = curr->next;
    }
}
```

Figure 6: find() and findAndUpdateStatus() function

Find(): This function create an element and make it as current node. if the doubly linked list is not empty then ,the element will then compare with the current node to validate. In this case, username, password and status would be validate. If the user username and password is exist and the status is true which is an active user then will return a true Boolean. Otherwise , the function will return a false.

findAndUpdateStatus(): This function allow user to search for the specific string id and update its status. It will first convert the string id to an integer. If the node is discovered , the function will updates its status to the status parameter given. Once the function find and update the status successfully , it will return a true Boolean.



```
int findLargestId()
{
    int largestId = head->data.getId();
    DoublyNode<T> *curr = head->next;

    while (curr != nullptr)
    {
        int currId = curr->data.getId();
        if (currId > largestId)
        {
            largestId = currId;
        }
        curr = curr->next;
    }
    return largestId;
}

template <class U>
void deleteNode(DoublyNode<U>* node)   //use to delete the tenant
{
    if (node == nullptr) {return;}
    if (node == head) {head = node->next;}
    if (node == tail) {tail = node->prev;}
    if (node->prev != nullptr) {node->prev->next = node->next;}
    if (node->next != nullptr) {node->next->prev = node->prev;}
    delete node;
    size--;
}
```

Figure 7: findLargetId() and deleteNode() function

findLargestId(): This function traverses the doubly linked list and finds the largest ID among all nodes. It initializes the largest ID with the ID of the head node and then iterates through the list, updating the largest ID whenever it finds a greater one. It returns the largest ID found.

deleteNode(): This template function takes a pointer to a node and deletes it from the doubly linked list. It handles various cases such as deleting the head or tail node, and appropriately updates the previous and next pointers of adjacent nodes. After deleting the node, it decreases the size of the list by one.

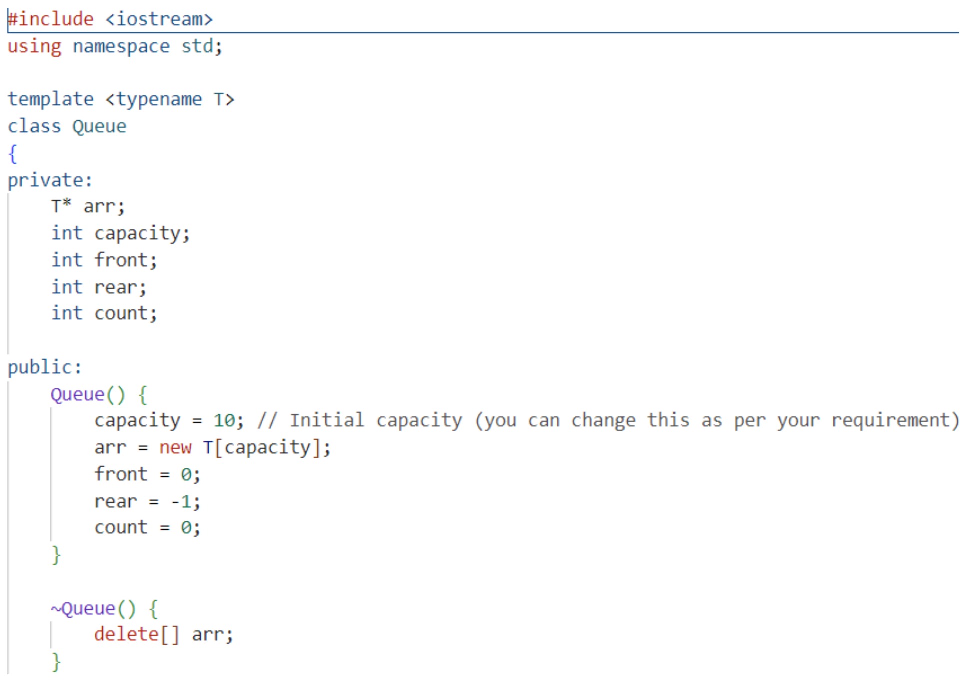
1.2.2 Queue

The queue header file helps define the functionality of a queue data structure, which is one of the fundamental data structure to be used in computer programming. A queue data structure follows the principles of First-In-First-Out, meaning that the elements added to the queue first will also be removed first.

The header information at the top of the queue header file contains essential information such as tenant's name, the date the code was written, and a brief description of the purpose of the code. This helps document the code and provide context.

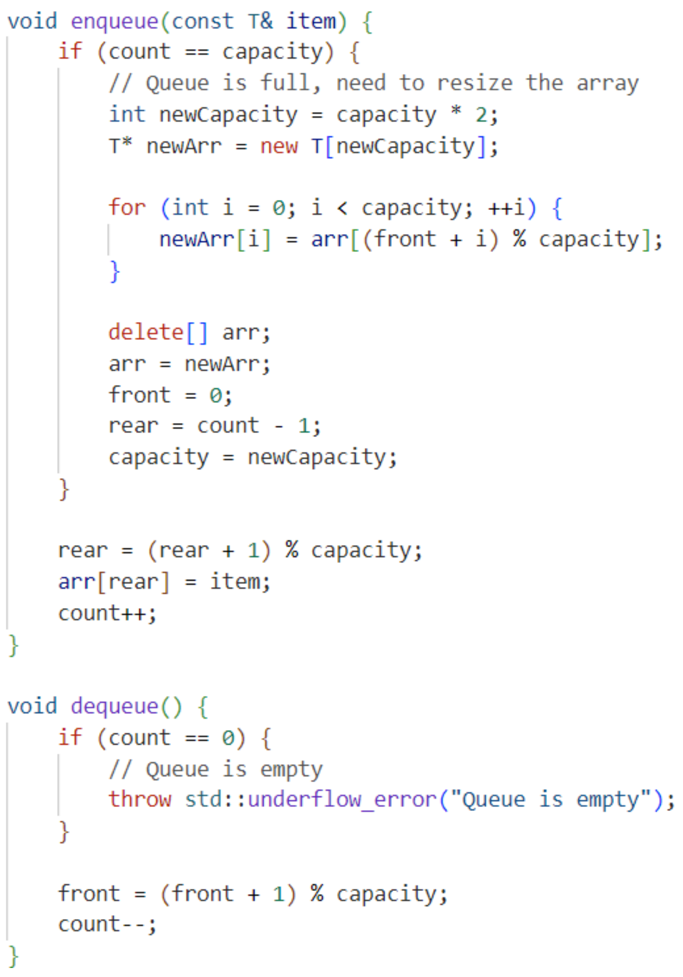
The code also includes preprocessor directives (ifndef, endif) to prevent the header from being included more than once in the same program. This is to avoid potential compilation errors that can arise from including the same header multiple times.

The queue header file defines a 'Queue' class. The class contains private variables such as 'capacity' which is an integer storing the maximum capacity of the queue, 'front' which is an integer indicating the front element's index, 'rear' which is an integer indicating the rear element's index and 'size' which is an integer representing the current size of the queue. The class contains public functions such as 'Queue()' which is a constructor that initializes the queue's capacity and sets the front, rear, and size, '~Queue()' which is a destructor that deallocates any memory allocated for queue elements, ~enqueue(element)' which is a function that adds an element to the rear of the queue, 'dequeue()' which is a function that dequeues/ removes the front element from the queue, 'front()' which is a function that retrieves the front element without dequeuing it, 'rear()' which is a function that front retrieves the rear element.



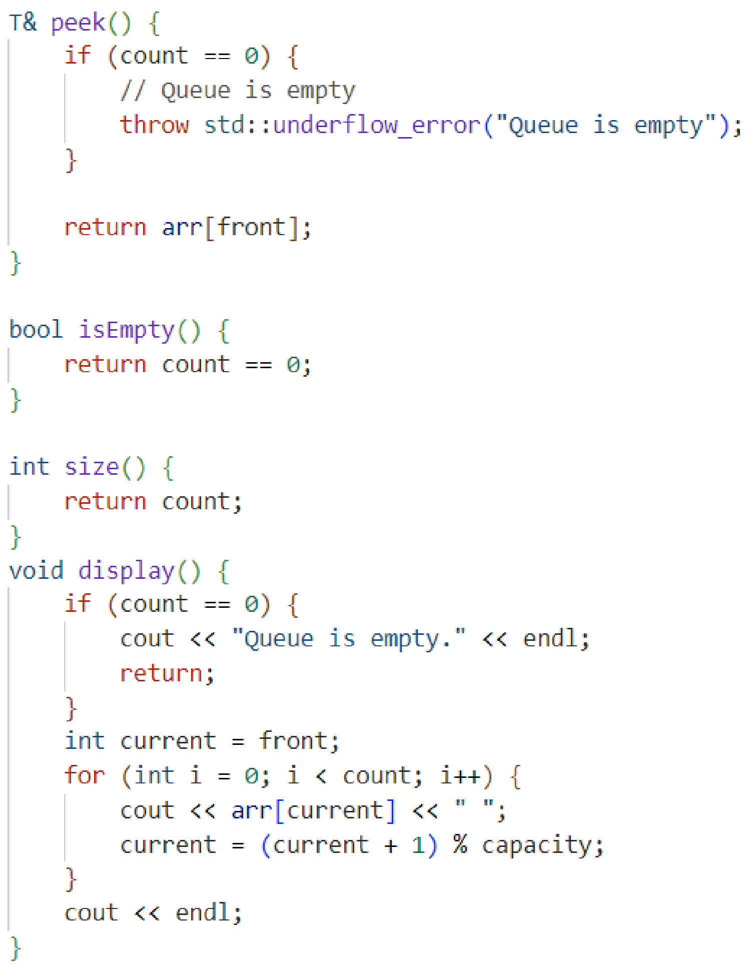
```
#include <iostream>
using namespace std;

template <typename T>
class Queue
{
private:
    T* arr;
    int capacity;
    int front;
    int rear;
    int count;

public:
    Queue() {
        capacity = 10; // Initial capacity (you can change this as per your requirement)
        arr = new T[capacity];
        front = 0;
        rear = -1;
        count = 0;
    }

    ~Queue() {
        delete[] arr;
    }
```

Figure Queue variables, constructor and destructor



```cpp
void enqueue(const T& item) {
    if (count == capacity) {
        // Queue is full, need to resize the array
        int newCapacity = capacity * 2;
        T* newArr = new T[newCapacity];

        for (int i = 0; i < capacity; ++i) {
            newArr[i] = arr[(front + i) % capacity];
        }

        delete[] arr;
        arr = newArr;
        front = 0;
        rear = count - 1;
        capacity = newCapacity;
    }

    rear = (rear + 1) % capacity;
    arr[rear] = item;
    count++;
}

void dequeue() {
    if (count == 0) {
        // Queue is empty
        throw std::underflow_error("Queue is empty");
    }

    front = (front + 1) % capacity;
    count--;
}
```

Figure 8: Queue enqueue function and dequeue function

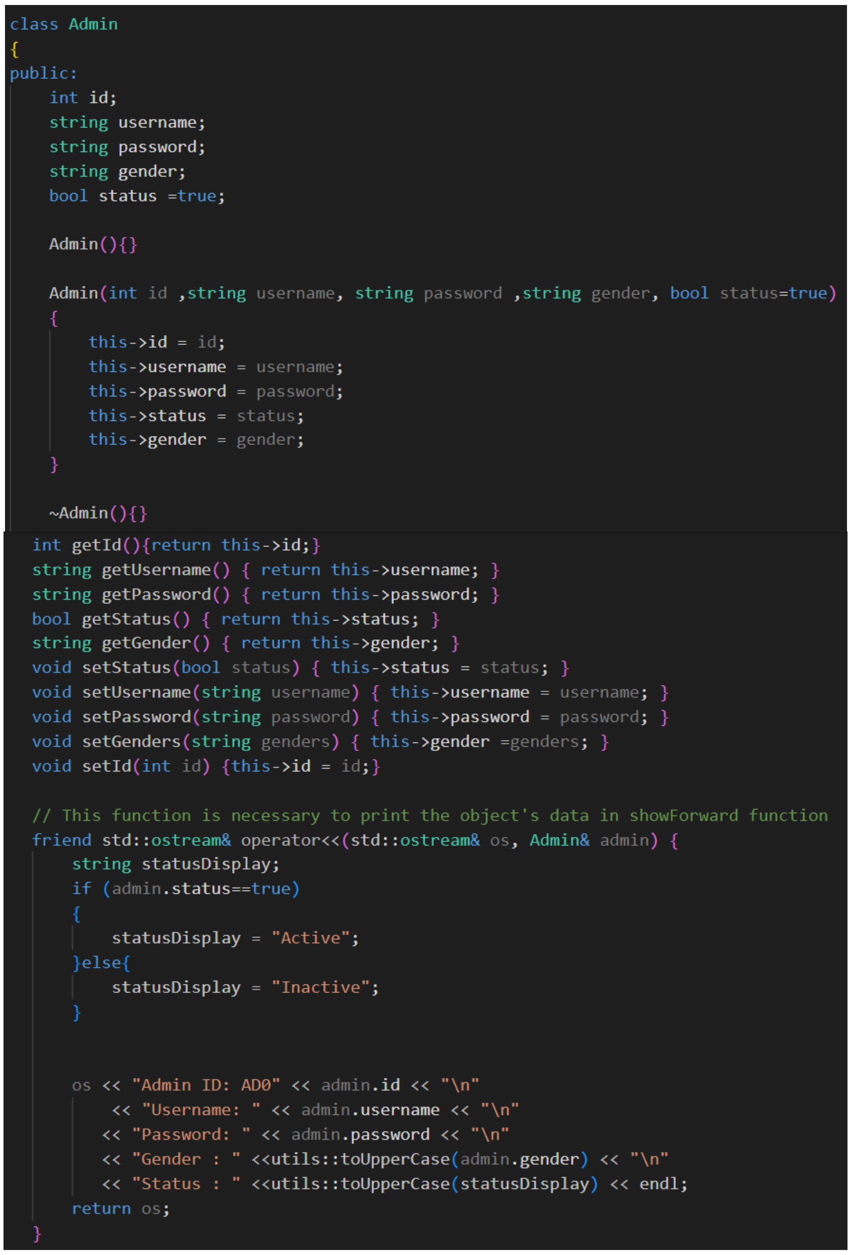


```
T& peek() {
    if (count == 0) {
        // Queue is empty
        throw std::underflow_error("Queue is empty");
    }

    return arr[front];
}

bool isEmpty() {
    return count == 0;
}

int size() {
    return count;
}
void display() {
    if (count == 0) {
        cout << "Queue is empty." << endl;
        return;
    }
    int current = front;
    for (int i = 0; i < count; i++) {
        cout << arr[current] << " ";
        current = (current + 1) % capacity;
    }
    cout << endl;
}
```

Figure 9: Queue peek function, isEmpty, size, display function

1.3 Class

1.3.1 Admin class

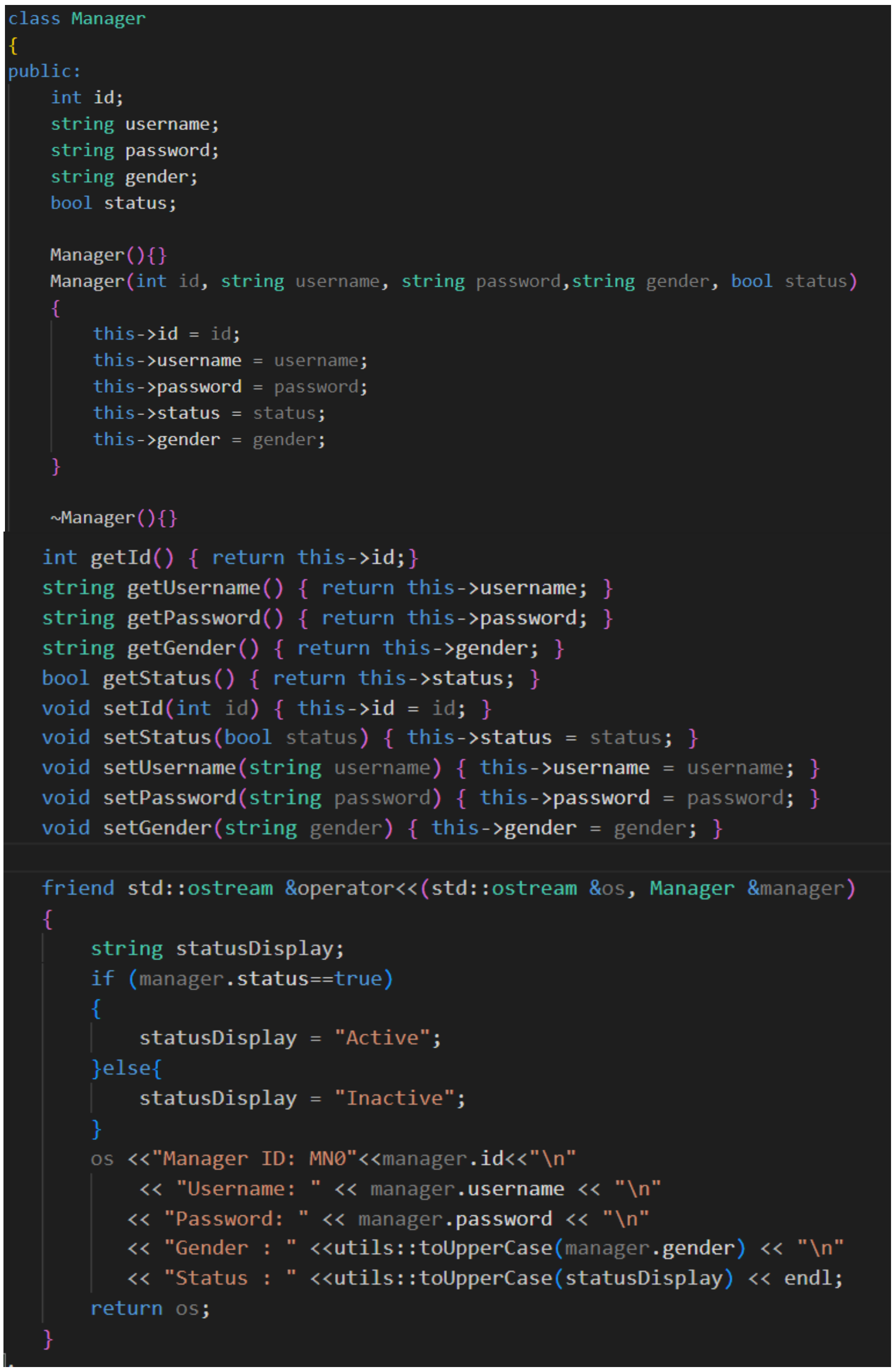


```cpp
class Admin
{
public:
    int id;
    string username;
    string password;
    string gender;
    bool status =true;

    Admin(){}

    Admin(int id ,string username, string password ,string gender, bool status=true)
    {
        this->id = id;
        this->username = username;
        this->password = password;
        this->status = status;
        this->gender = gender;
    }

    ~Admin(){}
  int getId(){return this->id;}
  string getUsername() { return this->username; }
  string getPassword() { return this->password; }
  bool getStatus() { return this->status; }
  string getGender() { return this->gender; }
  void setStatus(bool status) { this->status = status; }
  void setUsername(string username) { this->username = username; }
  void setPassword(string password) { this->password = password; }
  void setGenders(string genders) { this->gender =genders; }
  void setId(int id) {this->id = id;}

  // This function is necessary to print the object's data in showForward function
  friend std::ostream& operator<<(std::ostream& os, Admin& admin) {
      string statusDisplay;
      if (admin.status==true)
      {
          statusDisplay = "Active";
      }else{
          statusDisplay = "Inactive";
      }


      os << "Admin ID: AD0" << admin.id << "\n"
          << "Username: " << admin.username << "\n"
         << "Password: " << admin.password << "\n"
         << "Gender : " <<utils::toUpperCase(admin.gender) << "\n"
         << "Status : " <<utils::toUpperCase(statusDisplay) << endl;
      return os;
  }
```

Figure 10: Admin class

Diagram above are the admin class. There are 5 public variable has been created which is the id, username, password, status and gender. There are a constructor and destructor exist in the admin class to allow user to create admin and also let the program clean and release the admin data after the program execute. Moreover, getter and setter allow user to control the object data and increase the flexibility of the code. Next , we have define an overload insertion operator for the admin class. It will return the information set above when returning the stream.

1.3.2 Manager class



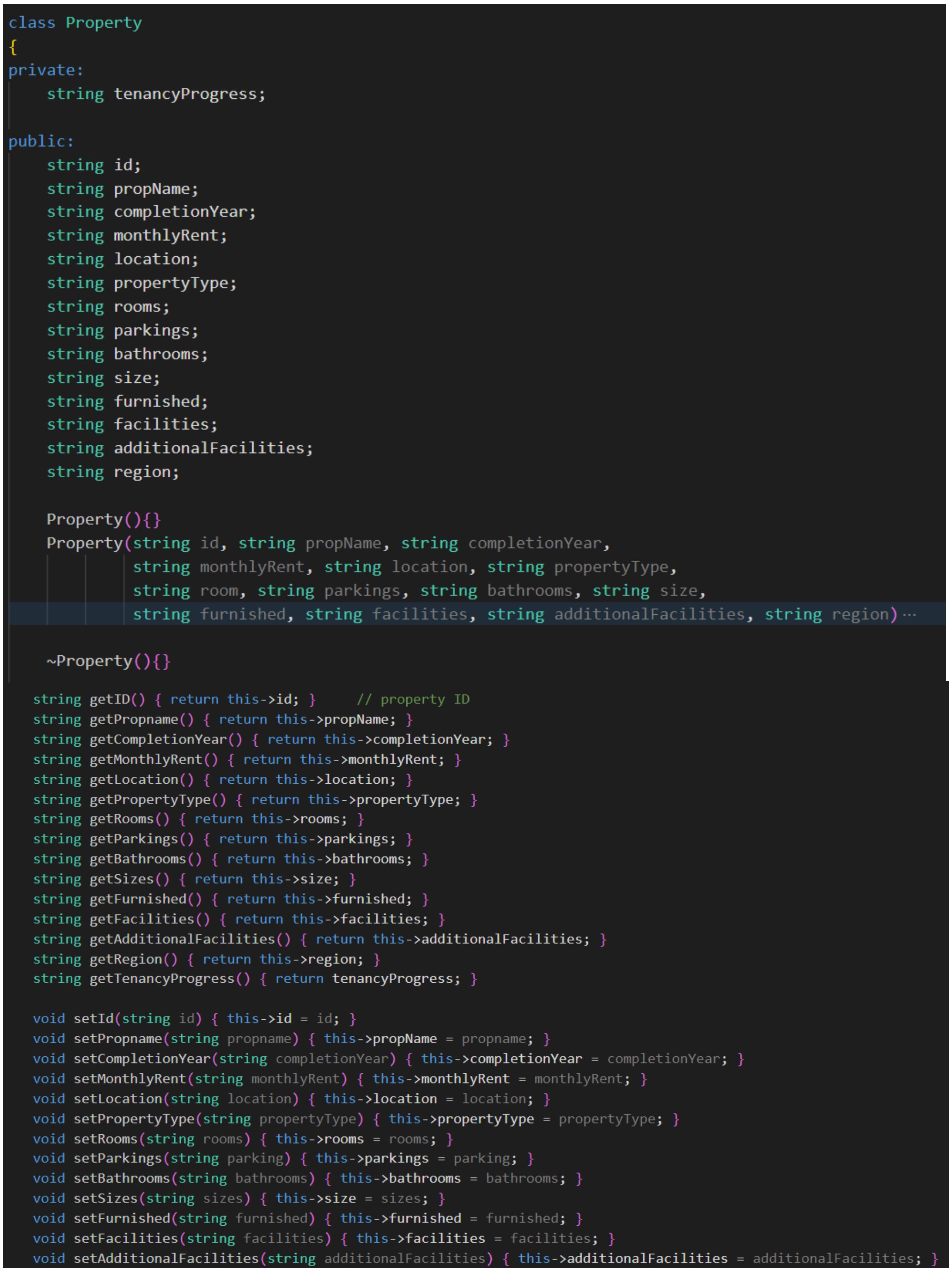
```
class Manager
{
public:
    int id;
    string username;
    string password;
    string gender;
    bool status;

    Manager(){}
    Manager(int id, string username, string password,string gender, bool status)
    {
        this->id = id;
        this->username = username;
        this->password = password;
        this->status = status;
        this->gender = gender;
    }

    ~Manager(){}
    int getId() { return this->id;}
    string getUsername() { return this->username; }
    string getPassword() { return this->password; }
    string getGender() { return this->gender; }
    bool getStatus() { return this->status; }
    void setId(int id) { this->id = id; }
    void setStatus(bool status) { this->status = status; }
    void setUsername(string username) { this->username = username; }
    void setPassword(string password) { this->password = password; }
    void setGender(string gender) { this->gender = gender; }

    friend std::ostream &operator<<(std::ostream &os, Manager &manager)
    {
        string statusDisplay;
        if (manager.status==true)
        {
            statusDisplay = "Active";
        }else{
            statusDisplay = "Inactive";
        }
        os <<"Manager ID: MN0"<<manager.id<<"\n"
            << "Username: " << manager.username << "\n"
           << "Password: " << manager.password << "\n"
           << "Gender : " <<utils::toUpperCase(manager.gender) << "\n"
           << "Status : " <<utils::toUpperCase(statusDisplay) << endl;
        return os;
    }
```

Figure 11: Manager class

Diagram above are the manager class. There are 5 public variable has been created which is the id, username, password, status and gender. There are a constructor and destructor exist in the manager class to allow user to create admin and also let the program clean and release the admin data after the program execute. Moreover, getter and setter allow user to control the object data and increase the flexibility of the code.Basicaaly , the manager class is similar with the admin class the only difference is the status. We has convert the type of the status from

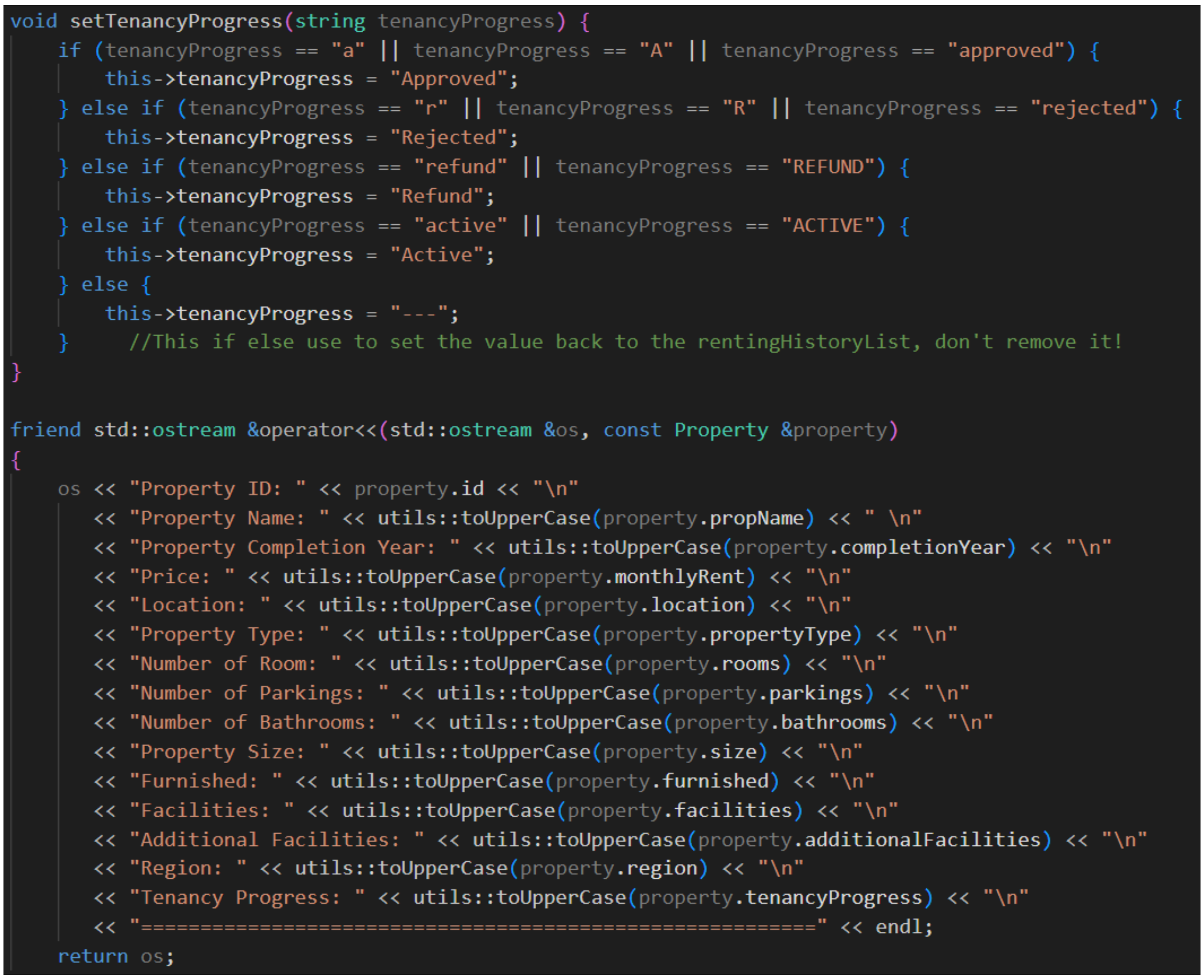
Boolean to string in order to display in the screen. Next , we have define an overload insertion operator for the admin class. It will return the information set above when returning the stream.

1.3.3 Property



```cpp
class Property
{
private:
    string tenancyProgress;

public:
    string id;
    string propName;
    string completionYear;
    string monthlyRent;
    string location;
    string propertyType;
    string rooms;
    string parkings;
    string bathrooms;
    string size;
    string furnished;
    string facilities;
    string additionalFacilities;
    string region;

    Property(){}
    Property(string id, string propName, string completionYear,
             string monthlyRent, string location, string propertyType,
             string room, string parkings, string bathrooms, string size,
             string furnished, string facilities, string additionalFacilities, string region)···

    ~Property(){}

    string getID() { return this->id; }     // property ID
    string getPropname() { return this->propName; }
    string getCompletionYear() { return this->completionYear; }
    string getMonthlyRent() { return this->monthlyRent; }
    string getLocation() { return this->location; }
    string getPropertyType() { return this->propertyType; }
    string getRooms() { return this->rooms; }
    string getParkings() { return this->parkings; }
    string getBathrooms() { return this->bathrooms; }
    string getSizes() { return this->size; }
    string getFurnished() { return this->furnished; }
    string getFacilities() { return this->facilities; }
    string getAdditionalFacilities() { return this->additionalFacilities; }
    string getRegion() { return this->region; }
    string getTenancyProgress() { return tenancyProgress; }

    void setId(string id) { this->id = id; }
    void setPropname(string propname) { this->propName = propname; }
    void setCompletionYear(string completionYear) { this->completionYear = completionYear; }
    void setMonthlyRent(string monthlyRent) { this->monthlyRent = monthlyRent; }
    void setLocation(string location) { this->location = location; }
    void setPropertyType(string propertyType) { this->propertyType = propertyType; }
    void setRooms(string rooms) { this->rooms = rooms; }
    void setParkings(string parking) { this->parkings = parking; }
    void setBathrooms(string bathrooms) { this->bathrooms = bathrooms; }
    void setSizes(string sizes) { this->size = sizes; }
    void setFurnished(string furnished) { this->furnished = furnished; }
    void setFacilities(string facilities) { this->facilities = facilities; }
    void setAdditionalFacilities(string additionalFacilities) { this->additionalFacilities = additionalFacilities; }
```

Figure 12:Property class

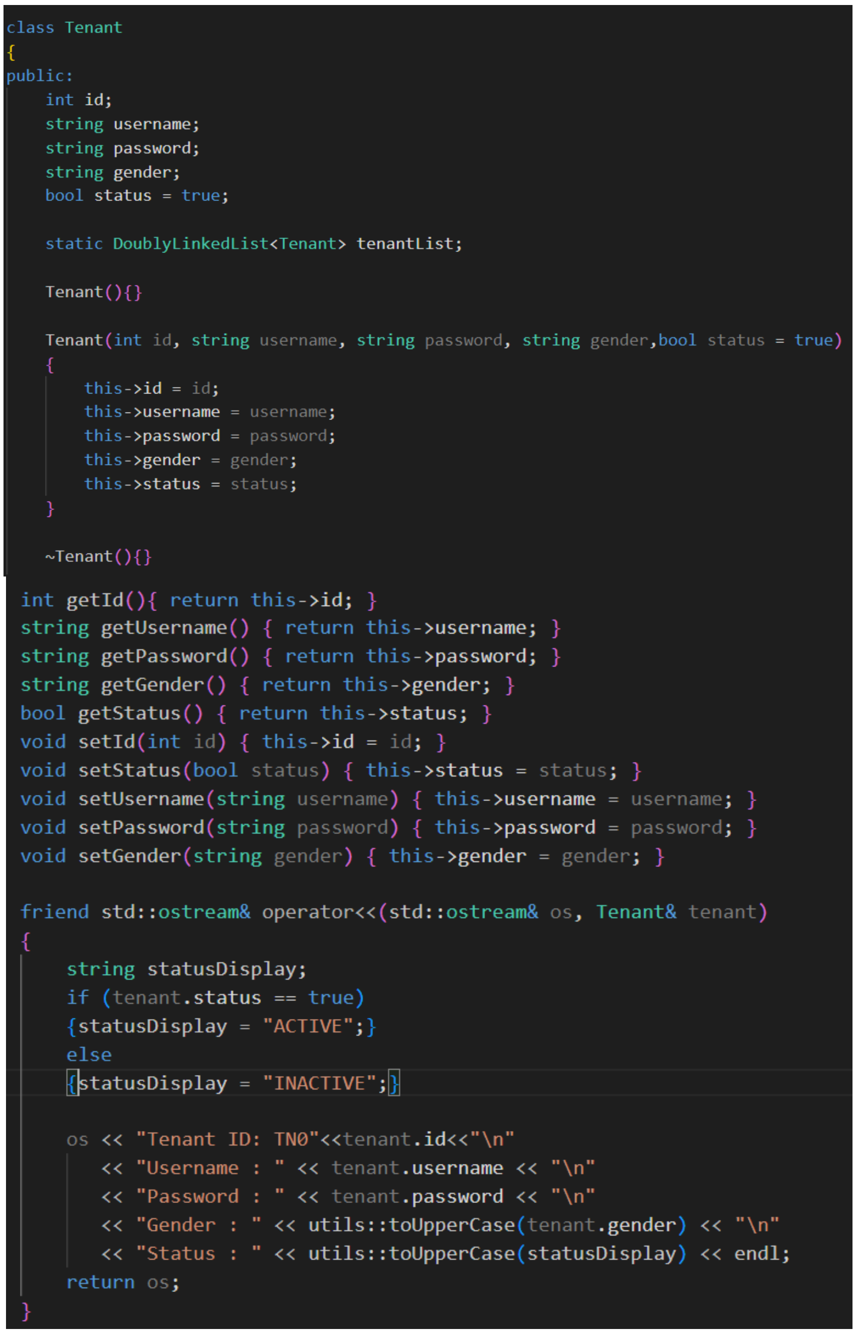


```
void setTenancyProgress(string tenancyProgress) {
    if (tenancyProgress == "a" || tenancyProgress == "A" || tenancyProgress == "approved") {
        this->tenancyProgress = "Approved";
    } else if (tenancyProgress == "r" || tenancyProgress == "R" || tenancyProgress == "rejected") {
        this->tenancyProgress = "Rejected";
    } else if (tenancyProgress == "refund" || tenancyProgress == "REFUND") {
        this->tenancyProgress = "Refund";
    } else if (tenancyProgress == "active" || tenancyProgress == "ACTIVE") {
        this->tenancyProgress = "Active";
    } else {
        this->tenancyProgress = "---";
    }   //This if else use to set the value back to the rentingHistoryList, don't remove it!
}

friend std::ostream &operator<<(std::ostream &os, const Property &property)
{
    os << "Property ID: " << property.id << "\n"
       << "Property Name: " << utils::toUpperCase(property.propName) << " \n"
       << "Property Completion Year: " << utils::toUpperCase(property.completionYear) << "\n"
       << "Price: " << utils::toUpperCase(property.monthlyRent) << "\n"
       << "Location: " << utils::toUpperCase(property.location) << "\n"
       << "Property Type: " << utils::toUpperCase(property.propertyType) << "\n"
       << "Number of Room: " << utils::toUpperCase(property.rooms) << "\n"
       << "Number of Parkings: " << utils::toUpperCase(property.parkings) << "\n"
       << "Number of Bathrooms: " << utils::toUpperCase(property.bathrooms) << "\n"
       << "Property Size: " << utils::toUpperCase(property.size) << "\n"
       << "Furnished: " << utils::toUpperCase(property.furnished) << "\n"
       << "Facilities: " << utils::toUpperCase(property.facilities) << "\n"
       << "Additional Facilities: " << utils::toUpperCase(property.additionalFacilities) << "\n"
       << "Region: " << utils::toUpperCase(property.region) << "\n"
       << "Tenancy Progress: " << utils::toUpperCase(property.tenancyProgress) << "\n"
       << "========================================================" << endl;
    return os;
```

Figure 13: Property Class

The Property class encapsulates information about a property, including various details like ID, name, completion year, monthly rent, location, property type, rooms, parking spaces, bathrooms, size, furnishings, facilities, additional facilities, and region. It provides constructors to initialize the values and a destructor. It also includes getter and setter methods to access and modify these attributes, with a specific logic implemented in setTenancyProgress() to manage different tenancy status. Furthermore, the class overloads the insertion operator, allowing for formatted output of a Property object, with details printed in an uppercase format using a utility function.

1.3.4 Tenant



```cpp
class Tenant
{
public:
    int id;
    string username;
    string password;
    string gender;
    bool status = true;

    static DoublyLinkedList<Tenant> tenantList;

    Tenant(){}

    Tenant(int id, string username, string password, string gender,bool status = true)
    {
        this->id = id;
        this->username = username;
        this->password = password;
        this->gender = gender;
        this->status = status;
    }

    ~Tenant(){}
    int getId(){ return this->id; }
    string getUsername() { return this->username; }
    string getPassword() { return this->password; }
    string getGender() { return this->gender; }
    bool getStatus() { return this->status; }
    void setId(int id) { this->id = id; }
    void setStatus(bool status) { this->status = status; }
    void setUsername(string username) { this->username = username; }
    void setPassword(string password) { this->password = password; }
    void setGender(string gender) { this->gender = gender; }

    friend std::ostream& operator<<(std::ostream& os, Tenant& tenant)
    {
        string statusDisplay;
        if (tenant.status == true)
        {statusDisplay = "ACTIVE";}
        else
        {statusDisplay = "INACTIVE";}

        os << "Tenant ID: TN0"<<tenant.id<<"\n"
           << "Username : " << tenant.username << "\n"
           << "Password : " << tenant.password << "\n"
           << "Gender : " << utils::toUpperCase(tenant.gender) << "\n"
           << "Status : " << utils::toUpperCase(statusDisplay) << endl;
        return os;
    }
```

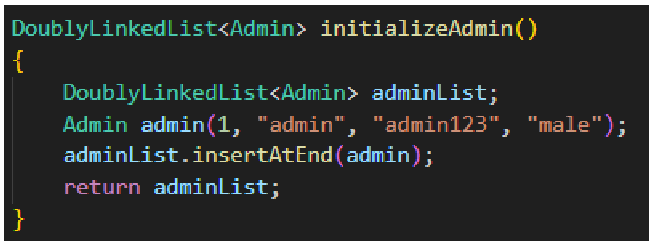
Figure 14: Tenant Class

The Tenant class represents a tenant with attributes such as ID, username, password, gender, and status, with the status indicating whether the tenant is active or inactive. The class provides constructors for initialization and destructors. It also includes getter and setter methods for accessing and modifying the attributes. Furthermore, the class overloads the insertion (<<) operator for formatted output, displaying the tenant details with status converted to a string as either "ACTIVE" or "INACTIVE". This class serves as a blueprint for creating and managing tenant information in the system.

1.4 Initialization of Data Structures

There are some functions that can be found which is for initializing various instances of the doubly linked list and queue data structures. These instances of the data structures will then be used in the system to store the types of data needed for the users to perform their functions in the system. There are also sample data included in the initialization process which help facilitate the user functions.

1.4.1 initializeAdmin

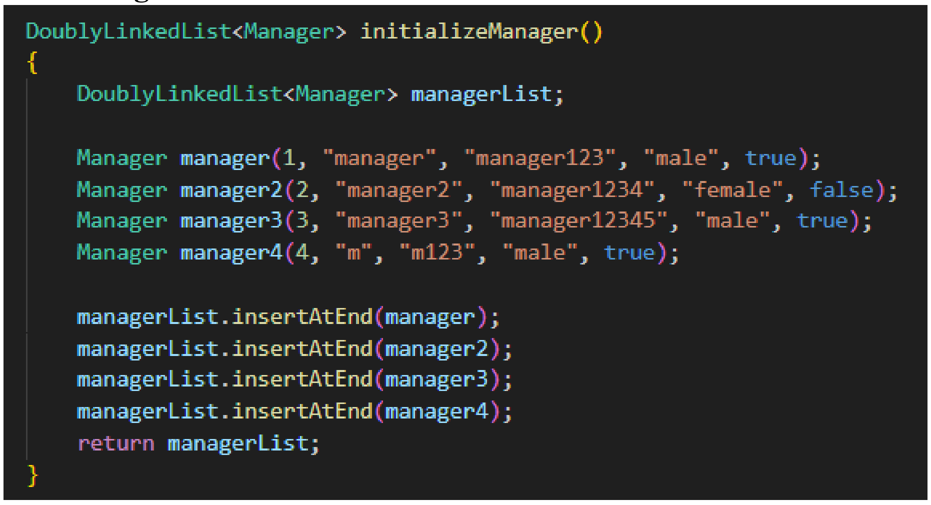


```
DoublyLinkedList<Admin> initializeAdmin()
{
    DoublyLinkedList<Admin> adminList;
    Admin admin(1, "admin", "admin123", "male");
    adminList.insertAtEnd(admin);
    return adminList;
}
```

Figure 15 initializeAdmin function

The adminList doubly linked list stores the admin's user details and is initialized with an admin user stored that has the id of 1, username "admin", password "admin123", and male gender. The admin object is first instantiated, which is then inserted into the doubly linked list data structure. The following initialization functions will follow a similar trend.

1.4.2 initializeManager

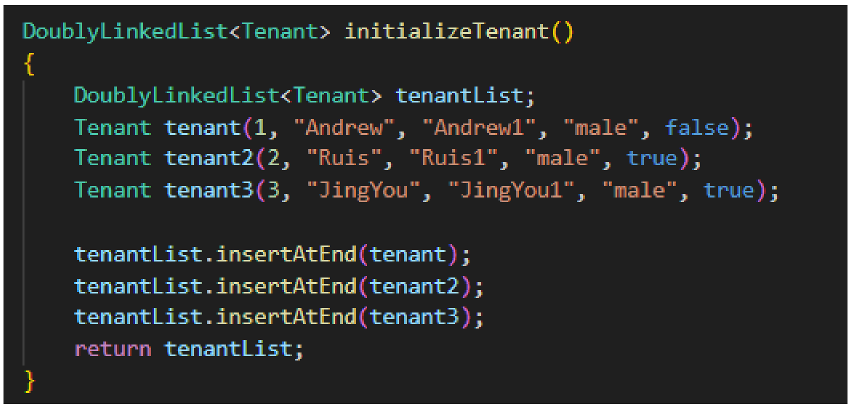


```
DoublyLinkedList<Manager> initializeManager()
{
    DoublyLinkedList<Manager> managerList;

    Manager manager(1, "manager", "manager123", "male", true);
    Manager manager2(2, "manager2", "manager1234", "female", false);
    Manager manager3(3, "manager3", "manager12345", "male", true);
    Manager manager4(4, "m", "m123", "male", true);

    managerList.insertAtEnd(manager);
    managerList.insertAtEnd(manager2);
    managerList.insertAtEnd(manager3);
    managerList.insertAtEnd(manager4);
    return managerList;
}
```

Figure 16 initializeManager

The managerList doubly linked list contains four sample manager users stored inside. The Boolean argument at the end of each record indicates the status of the manager, which is either "ACTIVE" or "INACTIVE".

1.4.3 initializeTenant

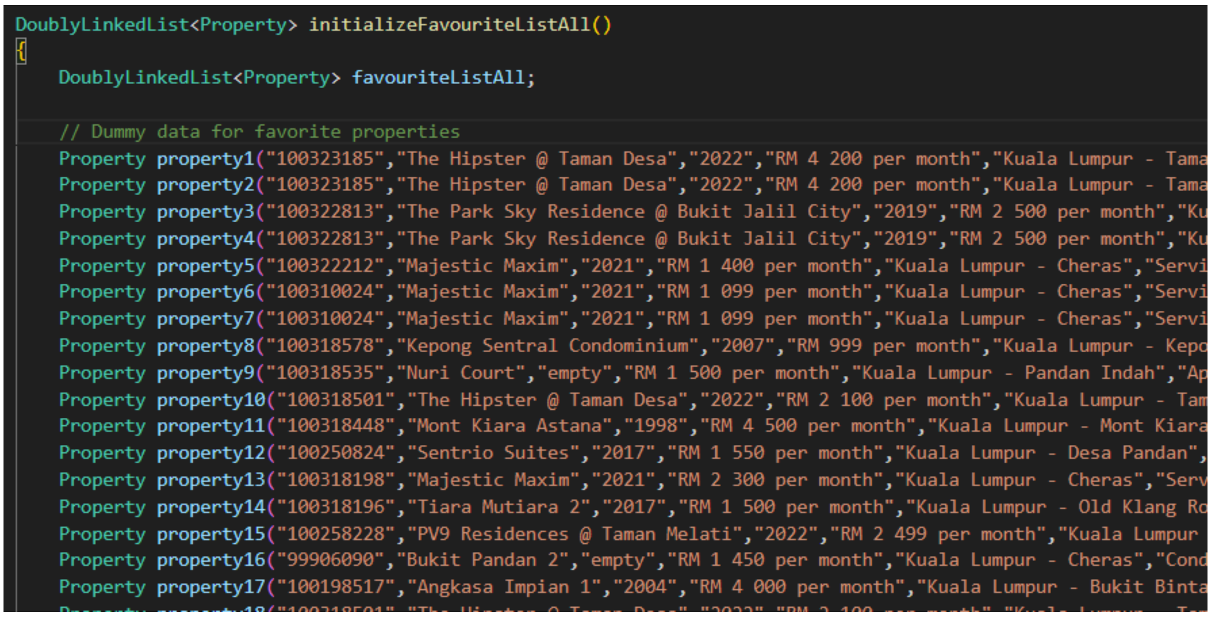


```
DoublyLinkedList<Tenant> initializeTenant()
{
    DoublyLinkedList<Tenant> tenantList;
    Tenant tenant(1, "Andrew", "Andrew1", "male", false);
    Tenant tenant2(2, "Ruis", "Ruis1", "male", true);
    Tenant tenant3(3, "JingYou", "JingYou1", "male", true);

    tenantList.insertAtEnd(tenant);
    tenantList.insertAtEnd(tenant2);
    tenantList.insertAtEnd(tenant3);
    return tenantList;
}
```

Figure 17 initializeTenant

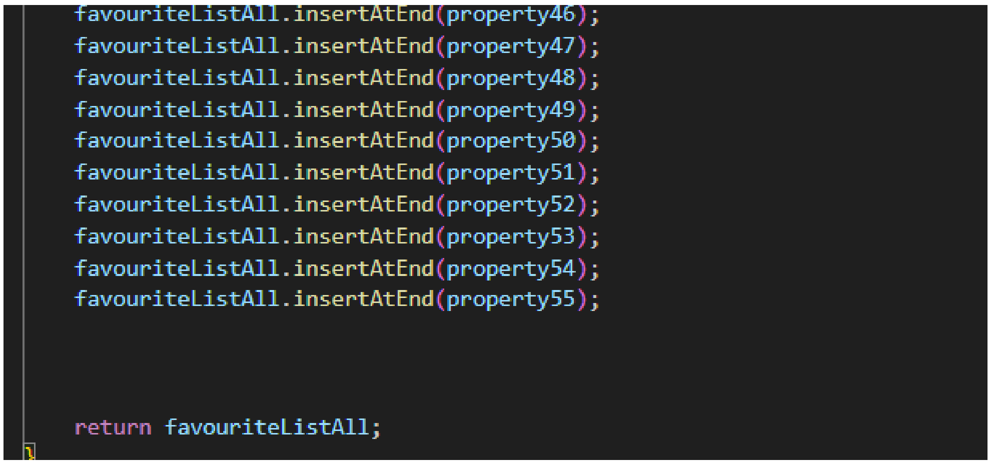
Like managerList, tenantList stores the user details of the tenants.

1.4.4 initializeFavouriteListAll



```
DoublyLinkedList<Property> initializeFavouriteListAll()
{
    DoublyLinkedList<Property> favouriteListAll;

    // Dummy data for favorite properties
    Property property1("100323185","The Hipster @ Taman Desa","2022","RM 4 200 per month","Kuala Lumpur - Tama
    Property property2("100323185","The Hipster @ Taman Desa","2022","RM 4 200 per month","Kuala Lumpur - Tama
    Property property3("100322813","The Park Sky Residence @ Bukit Jalil City","2019","RM 2 500 per month","Ku
    Property property4("100322813","The Park Sky Residence @ Bukit Jalil City","2019","RM 2 500 per month","Ku
    Property property5("100322212","Majestic Maxim","2021","RM 1 400 per month","Kuala Lumpur - Cheras","Servi
    Property property6("100310024","Majestic Maxim","2021","RM 1 099 per month","Kuala Lumpur - Cheras","Servi
    Property property7("100310024","Majestic Maxim","2021","RM 1 099 per month","Kuala Lumpur - Cheras","Servi
    Property property8("100318578","Kepong Sentral Condominium","2007","RM 999 per month","Kuala Lumpur - Kepo
    Property property9("100318535","Nuri Court","empty","RM 1 500 per month","Kuala Lumpur - Pandan Indah","Ap
    Property property10("100318501","The Hipster @ Taman Desa","2022","RM 2 100 per month","Kuala Lumpur - Tam
    Property property11("100318448","Mont Kiara Astana","1998","RM 4 500 per month","Kuala Lumpur - Mont Kiara
    Property property12("100250824","Sentrio Suites","2017","RM 1 550 per month","Kuala Lumpur - Desa Pandan",
    Property property13("100318198","Majestic Maxim","2021","RM 2 300 per month","Kuala Lumpur - Cheras","Serv
    Property property14("100318196","Tiara Mutiara 2","2017","RM 1 500 per month","Kuala Lumpur - Old Klang Ro
    Property property15("100258228","PV9 Residences @ Taman Melati","2022","RM 2 499 per month","Kuala Lumpur
    Property property16("99906090","Bukit Pandan 2","empty","RM 1 450 per month","Kuala Lumpur - Cheras","Cond
    Property property17("100198517","Angkasa Impian 1","2004","RM 4 000 per month","Kuala Lumpur - Bukit Binta
```

Figure 18 initializeFavouriteListAll (1 of 2)

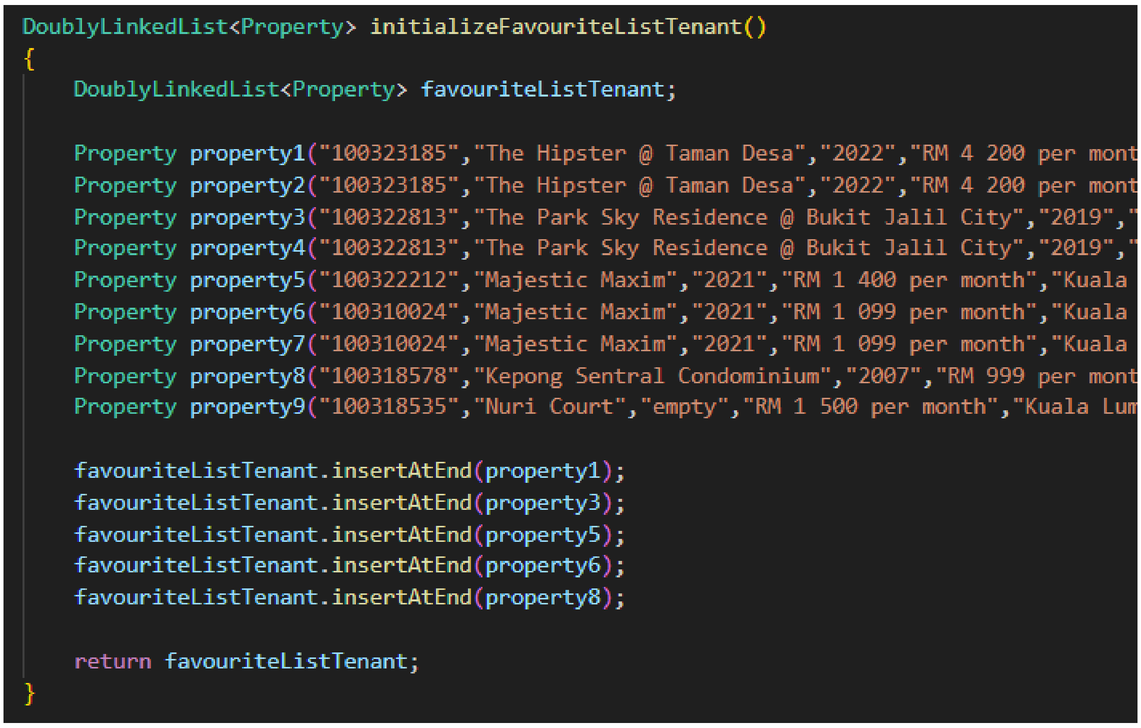


```
    favouriteListAll.insertAtEnd(property46);
    favouriteListAll.insertAtEnd(property47);
    favouriteListAll.insertAtEnd(property48);
    favouriteListAll.insertAtEnd(property49);
    favouriteListAll.insertAtEnd(property50);
    favouriteListAll.insertAtEnd(property51);
    favouriteListAll.insertAtEnd(property52);
    favouriteListAll.insertAtEnd(property53);
    favouriteListAll.insertAtEnd(property54);
    favouriteListAll.insertAtEnd(property55);



    return favouriteListAll;
}
```

Figure 19 initializeFavouriteListAll (2 of 2)

In the favouriteListAll doubly linked list, the records of a few properties are stored in a way that functions as a historical list of records. There are a total of 55 sample data stored during initialization.

1.4.5 initializeFavouriteListTenant

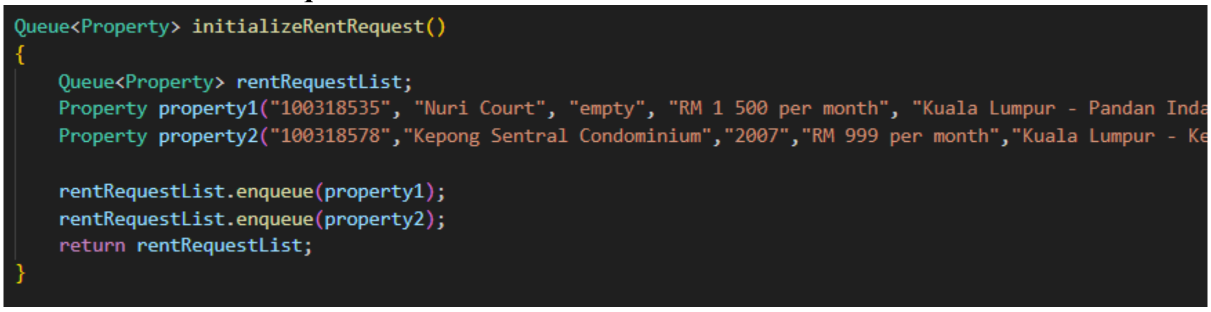


```
DoublyLinkedList<Property> initializeFavouriteListTenant()
{
    DoublyLinkedList<Property> favouriteListTenant;

    Property property1("100323185","The Hipster @ Taman Desa","2022","RM 4 200 per mont
    Property property2("100323185","The Hipster @ Taman Desa","2022","RM 4 200 per mont
    Property property3("100322813","The Park Sky Residence @ Bukit Jalil City","2019","
    Property property4("100322813","The Park Sky Residence @ Bukit Jalil City","2019","
    Property property5("100322212","Majestic Maxim","2021","RM 1 400 per month","Kuala
    Property property6("100310024","Majestic Maxim","2021","RM 1 099 per month","Kuala
    Property property7("100310024","Majestic Maxim","2021","RM 1 099 per month","Kuala
    Property property8("100318578","Kepong Sentral Condominium","2007","RM 999 per mont
    Property property9("100318535","Nuri Court","empty","RM 1 500 per month","Kuala Lum

    favouriteListTenant.insertAtEnd(property1);
    favouriteListTenant.insertAtEnd(property3);
    favouriteListTenant.insertAtEnd(property5);
    favouriteListTenant.insertAtEnd(property6);
    favouriteListTenant.insertAtEnd(property8);

    return favouriteListTenant;
}
```

Figure 20 initializeFavouriteListTenant

Similar to favouritePropertyAll, favouritePropertyTenant is used to store property details, but is initialized in a way that does not allow property with duplicate property ID to exist.

1.4.6 initializeRentRequest

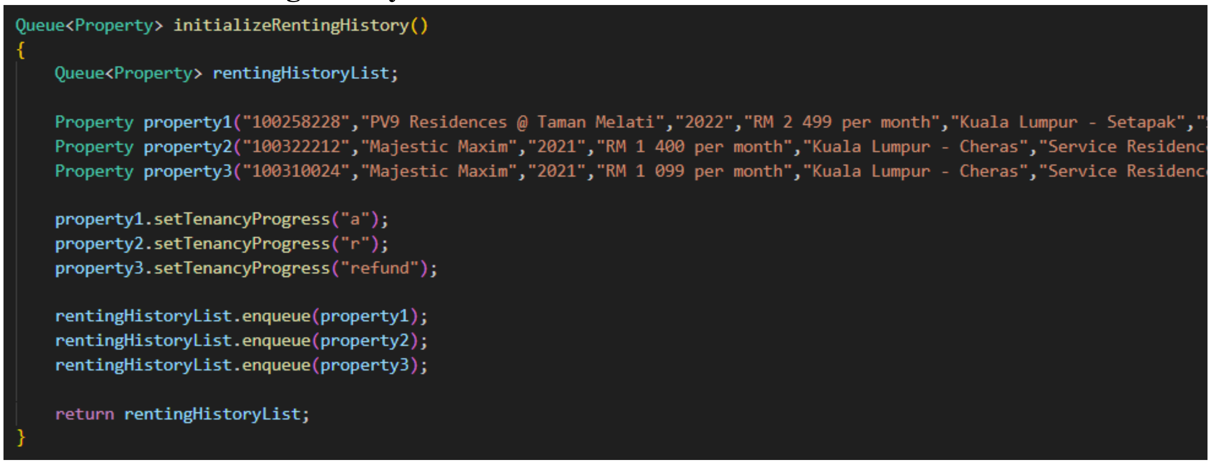


```
Queue<Property> initializeRentRequest()
{
    Queue<Property> rentRequestList;
    Property property1("100318535", "Nuri Court", "empty", "RM 1 500 per month", "Kuala Lumpur - Pandan Inda
    Property property2("100318578","Kepong Sentral Condominium","2007","RM 999 per month","Kuala Lumpur - Ke

    rentRequestList.enqueue(property1);
    rentRequestList.enqueue(property2);
    return rentRequestList;
}
```

Figure 21: initializeRentRequest

For rentRequestList, it is initialized as a queue data structure that stores property details which are added by tenants who wish to request to rent the property.

1.4.7 initializeRentingHistory

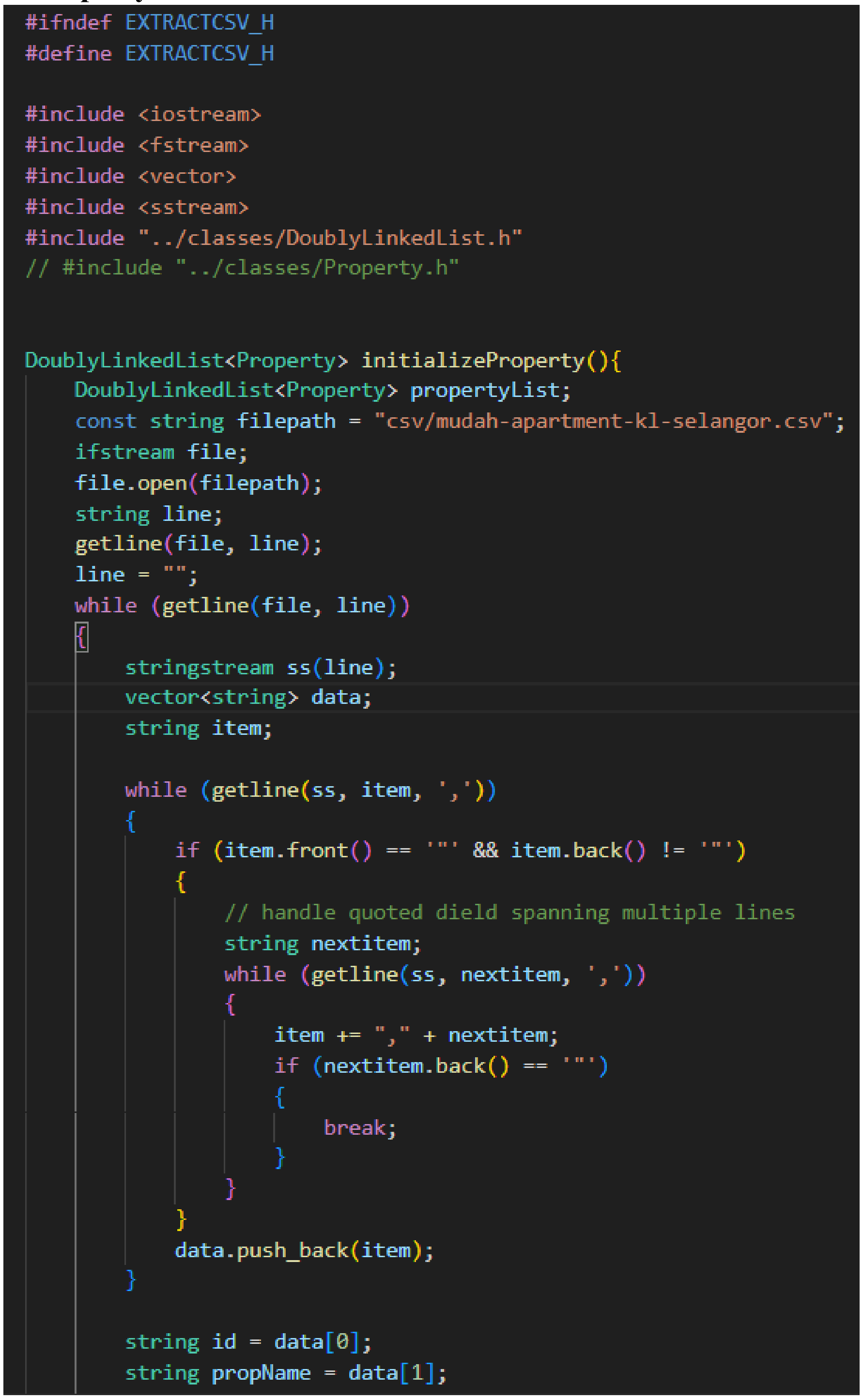


```
Queue<Property> initializeRentingHistory()
{
    Queue<Property> rentingHistoryList;

    Property property1("100258228","PV9 Residences @ Taman Melati","2022","RM 2 499 per month","Kuala Lumpur - Setapak","
    Property property2("100322212","Majestic Maxim","2021","RM 1 400 per month","Kuala Lumpur - Cheras","Service Residenc
    Property property3("100310024","Majestic Maxim","2021","RM 1 099 per month","Kuala Lumpur - Cheras","Service Residenc

    property1.setTenancyProgress("a");
    property2.setTenancyProgress("r");
    property3.setTenancyProgress("refund");

    rentingHistoryList.enqueue(property1);
    rentingHistoryList.enqueue(property2);
    rentingHistoryList.enqueue(property3);

    return rentingHistoryList;
}
```

Figure 22 :initializeRentingHistory

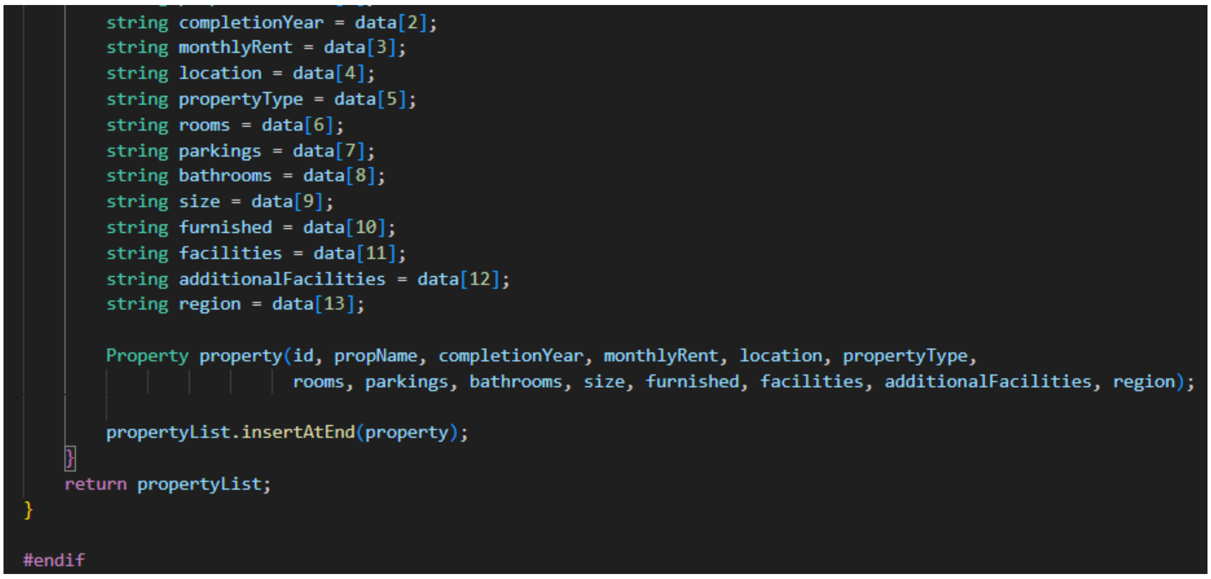
The rentingHistoryList is another queue data structure which stores the historical records of properties, with the added step that includes the tenancy progress of the property, which indicates the property to be “Accepted”, “Rejected”, “Refunded”.

1.4.8 initializeProperty



```cpp
#ifndef EXTRACTCSV_H
#define EXTRACTCSV_H

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include "../classes/DoublyLinkedList.h"
// #include "../classes/Property.h"


DoublyLinkedList<Property> initializeProperty(){
    DoublyLinkedList<Property> propertyList;
    const string filepath = "csv/mudah-apartment-kl-selangor.csv";
    ifstream file;
    file.open(filepath);
    string line;
    getline(file, line);
    line = "";
    while (getline(file, line))
    {
        stringstream ss(line);
        vector<string> data;
        string item;

        while (getline(ss, item, ','))
        {
            if (item.front() == '"' && item.back() != '"')
            {
                // handle quoted dield spanning multiple lines
                string nextitem;
                while (getline(ss, nextitem, ','))
                {
                    item += "," + nextitem;
                    if (nextitem.back() == '"')
                    {
                        break;
                    }
                }
            }
            data.push_back(item);
        }

        string id = data[0];
        string propName = data[1];
```

Figure 23: initializeProperty (1 of 2)



```cpp
        string completionYear = data[2];
        string monthlyRent = data[3];
        string location = data[4];
        string propertyType = data[5];
        string rooms = data[6];
        string parkings = data[7];
        string bathrooms = data[8];
        string size = data[9];
        string furnished = data[10];
        string facilities = data[11];
        string additionalFacilities = data[12];
        string region = data[13];

        Property property(id, propName, completionYear, monthlyRent, location, propertyType,
                          rooms, parkings, bathrooms, size, furnished, facilities, additionalFacilities, region);

        propertyList.insertAtEnd(property);
    }
    return propertyList;
}

#endif
```

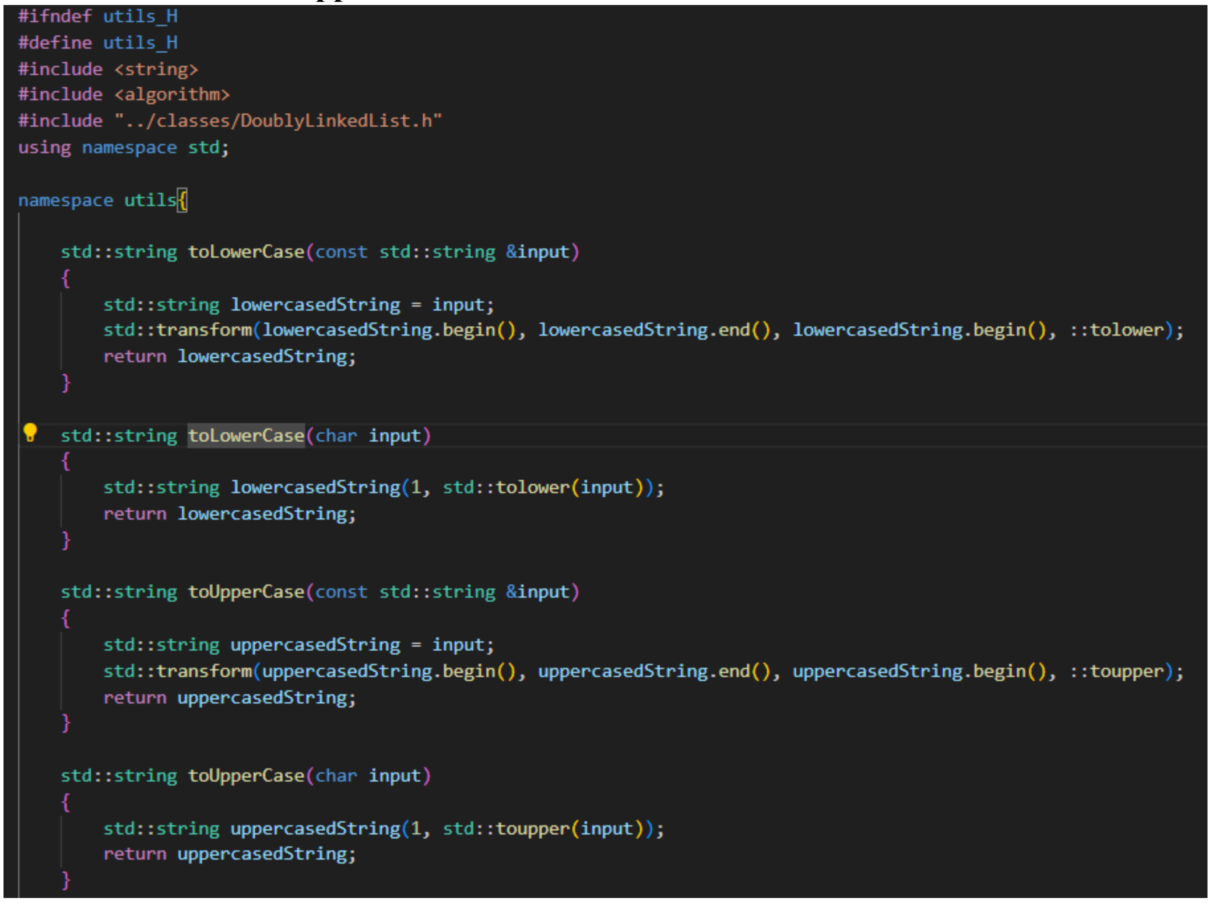
Figure 24: initializeProperty (2 of 2)

For the initialization of propertyList, it uses the doubly linked list data structure, and the data is obtained from the provided csv file. This initialization function is separated into its own header file, named extractcsv.h, which allows the propertyList initialization function to be called later in the program.

1.5 Utils

Utils refer to the utils.h file, which contains a few utility functions that has a general use case which aims to simplify functions or provide data validation.

1.5.1 toLowerCase/toUpperCase

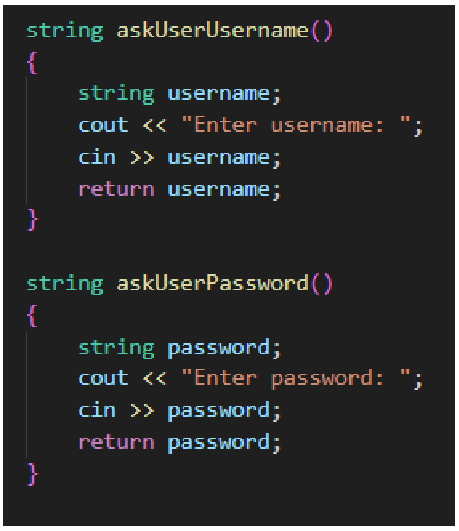


```cpp
#ifndef utils_H
#define utils_H
#include <string>
#include <algorithm>
#include "../classes/DoublyLinkedList.h"
using namespace std;

namespace utils{

    std::string toLowerCase(const std::string &input)
    {
        std::string lowercasedString = input;
        std::transform(lowercasedString.begin(), lowercasedString.end(), lowercasedString.begin(), ::tolower);
        return lowercasedString;
    }

    std::string toLowerCase(char input)
    {
        std::string lowercasedString(1, std::tolower(input));
        return lowercasedString;
    }

    std::string toUpperCase(const std::string &input)
    {
        std::string uppercasedString = input;
        std::transform(uppercasedString.begin(), uppercasedString.end(), uppercasedString.begin(), ::toupper);
        return uppercasedString;
    }

    std::string toUpperCase(char input)
    {
        std::string uppercasedString(1, std::toupper(input));
        return uppercasedString;
    }
```

Figure 25:toLowerCase/toUpperCase

Both toLowerCase and toUpperCase functions to convert string or char input into fully lowercase or uppercase respectively. This is used in the login section to provide a non-case specific checking for usernames and passwords in the login section.

1.5.2 askUserUsername/askUserPassword

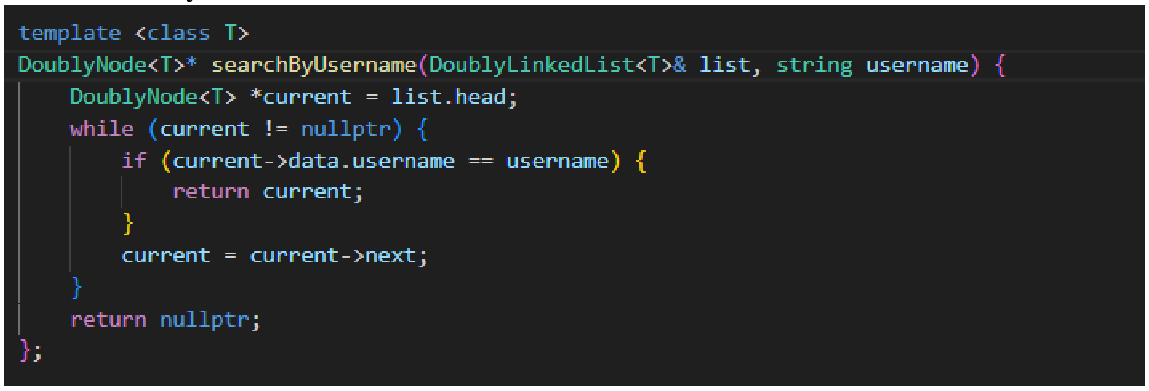


```
string askUserUsername()
{
    string username;
    cout << "Enter username: ";
    cin >> username;
    return username;
}

string askUserPassword()
{
    string password;
    cout << "Enter password: ";
    cin >> password;
    return password;
}
```

Figure 26 askUserUsername/askUserPassword

The askUserUsername and askUserPassword function is called to prompt the user to type in their usernames and password.

1.5.3 searchByUsername

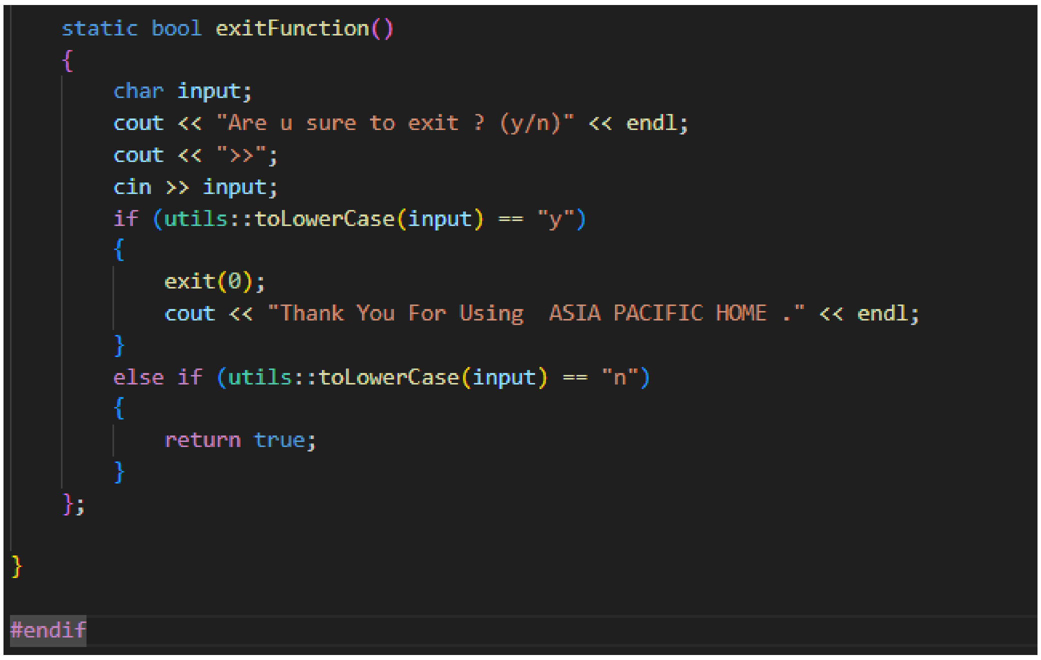


```
template <class T>
DoublyNode<T>* searchByUsername(DoublyLinkedList<T>& list, string username) {
    DoublyNode<T> *current = list.head;
    while (current != nullptr) {
        if (current->data.username == username) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
};
```

Figure 27 searchByUsername

The searchByUsername function can be called to loop through a doublylinkedlist data structure to return a node where the specified username is found.

1.5.4 exitFunction



```
    static bool exitFunction()
    {
        char input;
        cout << "Are u sure to exit ? (y/n)" << endl;
        cout << ">>";
        cin >> input;
        if (utils::toLowerCase(input) == "y")
        {
            exit(0);
            cout << "Thank You For Using  ASIA PACIFIC HOME ." << endl;
        }
        else if (utils::toLowerCase(input) == "n")
        {
            return true;
        }
    };

}

#endif
```

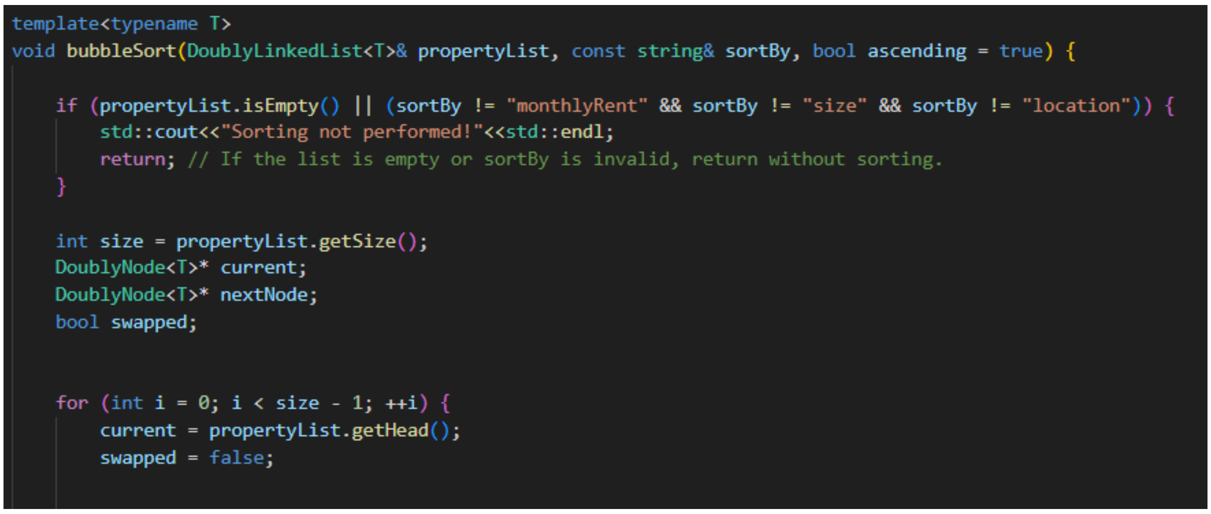
Figure 28 exitFunction

The exitFunction is called when user prompts to exit the system, which the user can then confirm to exit by typing “y”, or go back by typing “n”.

2.0 Implementation and Result

2.1 Source code for each algorithm

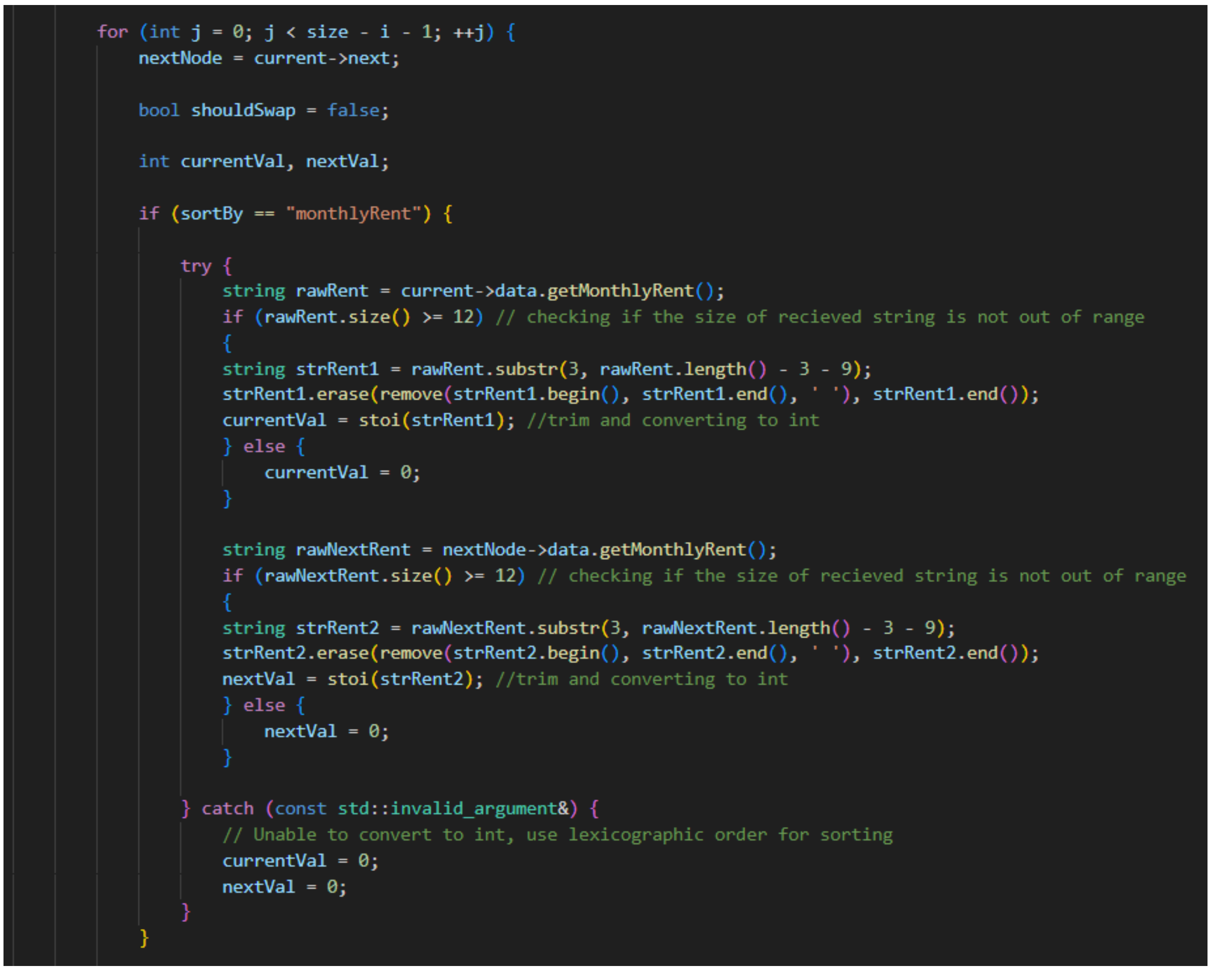
2.1.1 Bubble Sort



```
template<typename T>
void bubbleSort(DoublyLinkedList<T>& propertyList, const string& sortBy, bool ascending = true) {

    if (propertyList.isEmpty() || (sortBy != "monthlyRent" && sortBy != "size" && sortBy != "location")) {
        std::cout<<"Sorting not performed!"<<std::endl;
        return; // If the list is empty or sortBy is invalid, return without sorting.
    }

    int size = propertyList.getSize();
    DoublyNode<T>* current;
    DoublyNode<T>* nextNode;
    bool swapped;


    for (int i = 0; i < size - 1; ++i) {
        current = propertyList.getHead();
        swapped = false;
```

Figure 29 bubbleSort function (1 of 4)

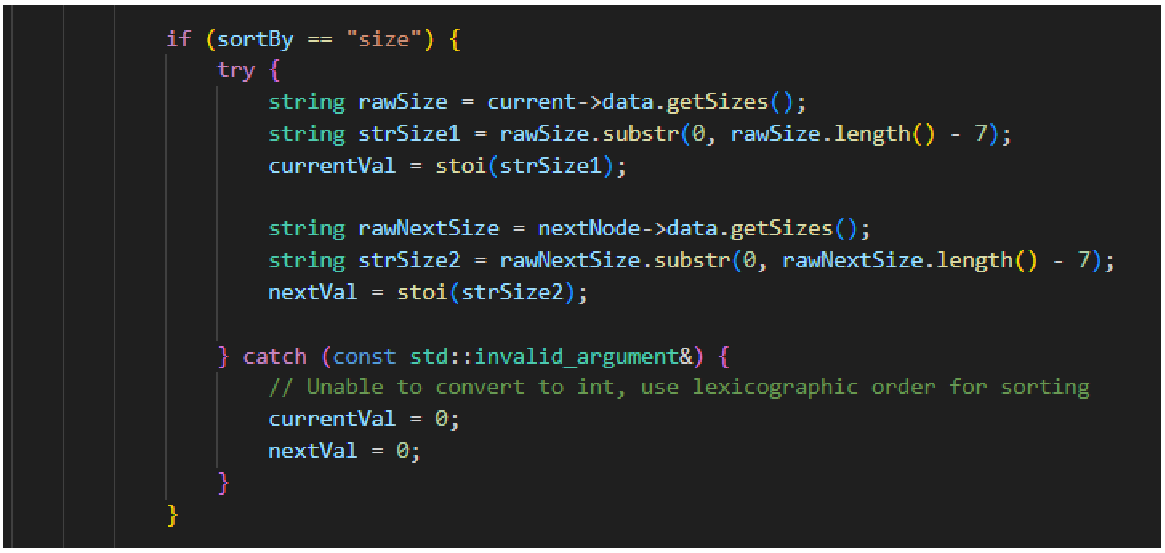
The bubbleSort function begins by passing in the propertylist, which characteristic of the property to sort by (monthly rent, size or location), and whether to sort the list in ascending order or descending order. It validates the arguments by checking if propertyList is empty, or if the sortBy arguments matches with either of the three characteristics. There is an outer for loop, which keeps track on the number of times traversed through the list, as well as the Boolean variable swapped, which indicates if the list is fully sorted.



```
        for (int j = 0; j < size - i - 1; ++j) {
            nextNode = current->next;

            bool shouldSwap = false;

            int currentVal, nextVal;

            if (sortBy == "monthlyRent") {

                try {
                    string rawRent = current->data.getMonthlyRent();
                    if (rawRent.size() >= 12) // checking if the size of recieved string is not out of range
                    {
                    string strRent1 = rawRent.substr(3, rawRent.length() - 3 - 9);
                    strRent1.erase(remove(strRent1.begin(), strRent1.end(), ' '), strRent1.end());
                    currentVal = stoi(strRent1); //trim and converting to int
                    } else {
                        currentVal = 0;
                    }

                    string rawNextRent = nextNode->data.getMonthlyRent();
                    if (rawNextRent.size() >= 12) // checking if the size of recieved string is not out of range
                    {
                    string strRent2 = rawNextRent.substr(3, rawNextRent.length() - 3 - 9);
                    strRent2.erase(remove(strRent2.begin(), strRent2.end(), ' '), strRent2.end());
                    nextVal = stoi(strRent2); //trim and converting to int
                    } else {
                        nextVal = 0;
                    }

                } catch (const std::invalid_argument&) {
                    // Unable to convert to int, use lexicographic order for sorting
                    currentVal = 0;
                    nextVal = 0;
                }
            }
```

Figure 30 bubbleSort function (2 of 4)

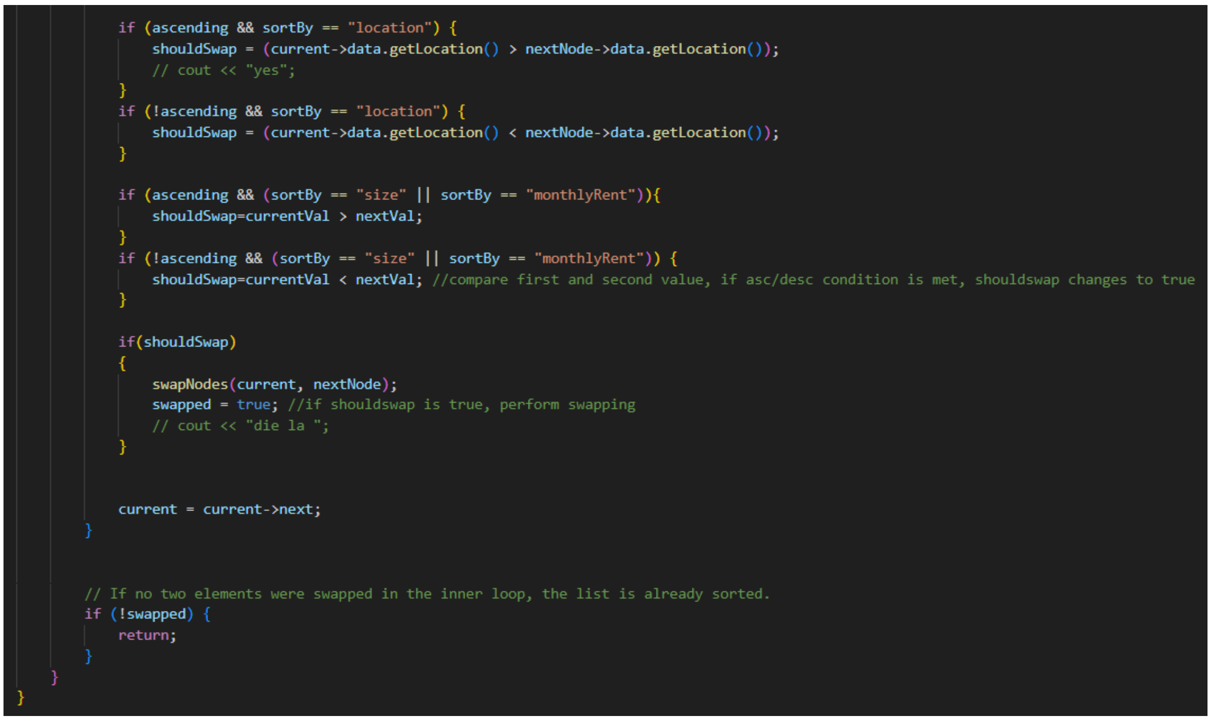
Moving into the inner loop, which is responsible for traversing through the propertyList as well as comparing the adjacent nodes. The Boolean variable shouldSwap is the indicator which decides if the two nodes being compared should be swapped so they are in order. There are also datatype conversions being done in the inner loop, as for data such as monthly rent or size, the data will have to be converted from their original type into integer, so that numerical comparisons between data is possible.



```cpp
            if (sortBy == "size") {
                try {
                    string rawSize = current->data.getSizes();
                    string strSize1 = rawSize.substr(0, rawSize.length() - 7);
                    currentVal = stoi(strSize1);

                    string rawNextSize = nextNode->data.getSizes();
                    string strSize2 = rawNextSize.substr(0, rawNextSize.length() - 7);
                    nextVal = stoi(strSize2);

                } catch (const std::invalid_argument&) {
                    // Unable to convert to int, use lexicographic order for sorting
                    currentVal = 0;
                    nextVal = 0;
                }
            }
```

Figure 31 bubbleSort function (3 of 4)

The size variable has a slightly different way than monthly rent in terms of trimming the data into a form that can be converted into integer.

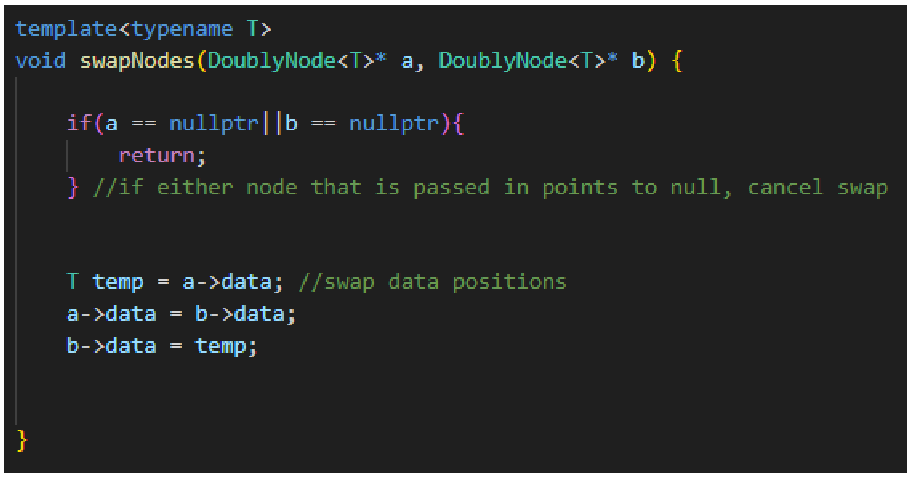


```cpp
            if (ascending && sortBy == "location") {
                shouldSwap = (current->data.getLocation() > nextNode->data.getLocation());
                // cout << "yes";
            }
            if (!ascending && sortBy == "location") {
                shouldSwap = (current->data.getLocation() < nextNode->data.getLocation());
            }

            if (ascending && (sortBy == "size" || sortBy == "monthlyRent")){
                shouldSwap=currentVal > nextVal;
            }
            if (!ascending && (sortBy == "size" || sortBy == "monthlyRent")) {
                shouldSwap=currentVal < nextVal; //compare first and second value, if asc/desc condition is met, shouldswap changes to true
            }

            if(shouldSwap)
            {
                swapNodes(current, nextNode);
                swapped = true; //if shouldswap is true, perform swapping
                // cout << "die la ";
            }


            current = current->next;
        }


        // If no two elements were swapped in the inner loop, the list is already sorted.
        if (!swapped) {
            return;
        }
    }
}
```

Figure 32 bubbleSort function (4 of 4)

After assigning the appropriate variable values into the currentVal and nextVal variables, they are compared in the if statements as shown, which decides if the two nodes will have to be swapped. Both loops proceeds until the “swapped” variable does not change to true, which completes the bubbleSort function.

The bubbleSort function is the core implementation of the proposed Bubble Sort algorithm. The bubble sort works by swapping the positions of adjacent elements given the order of sorting or if one is larger than the other. This simple process is repeated constantly until the propertyList is sorted completely.

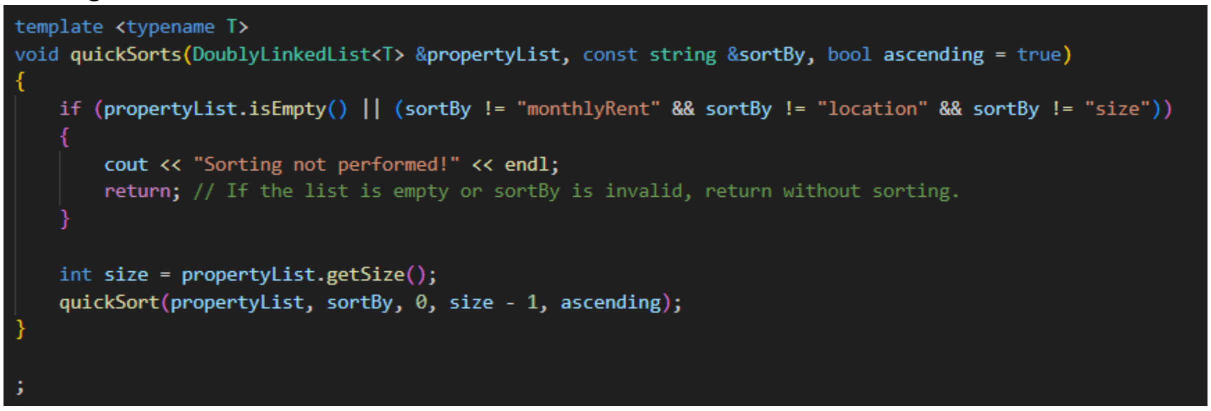


```
template<typename T>
void swapNodes(DoublyNode<T>* a, DoublyNode<T>* b) {

    if(a == nullptr||b == nullptr){
        return;
    } //if either node that is passed in points to null, cancel swap


    T temp = a->data; //swap data positions
    a->data = b->data;
    b->data = temp;


}
```

Figure 33 swapNode function

The swapnodes function is used for swapping the data of two nodes in the doubly linked list that is passed in through the arguments

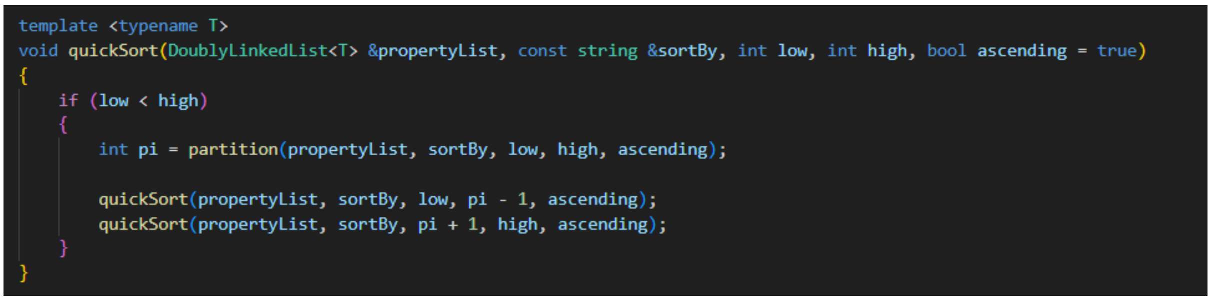
2.1.2 Quick Sort



```
template <typename T>
void quickSorts(DoublyLinkedList<T> &propertyList, const string &sortBy, bool ascending = true)
{
    if (propertyList.isEmpty() || (sortBy != "monthlyRent" && sortBy != "location" && sortBy != "size"))
    {
        cout << "Sorting not performed!" << endl;
        return; // If the list is empty or sortBy is invalid, return without sorting.
    }

    int size = propertyList.getSize();
    quickSort(propertyList, sortBy, 0, size - 1, ascending);
}

;
```

Figure 34 quickSorts function

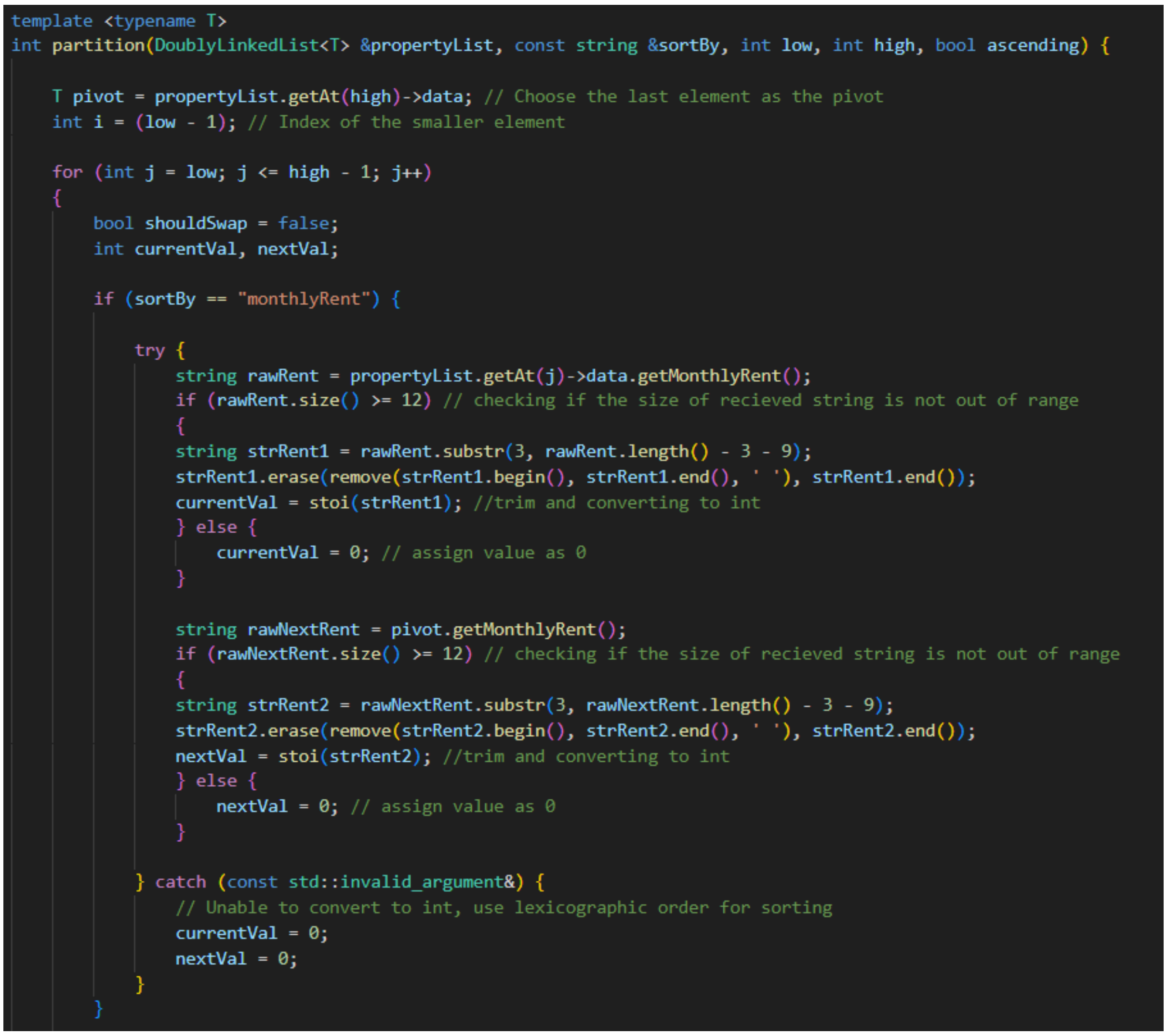
The quickSorts function where the Quick Sort algorithm starts, by passing the required arguments into this function, which are the propertyList, which data to sort by, and in ascending order or descending order. Like bubbleSort, it checks if the list is empty or if the sortBy value is valid before proceeding with the quick sort. This function assigns the first element of the propertyList as “low”, and the last element as “high”.



```
template <typename T>
void quickSort(DoublyLinkedList<T> &propertyList, const string &sortBy, int low, int high, bool ascending = true)
{
    if (low < high)
    {
        int pi = partition(propertyList, sortBy, low, high, ascending);

        quickSort(propertyList, sortBy, low, pi - 1, ascending);
        quickSort(propertyList, sortBy, pi + 1, high, ascending);
    }
}
```

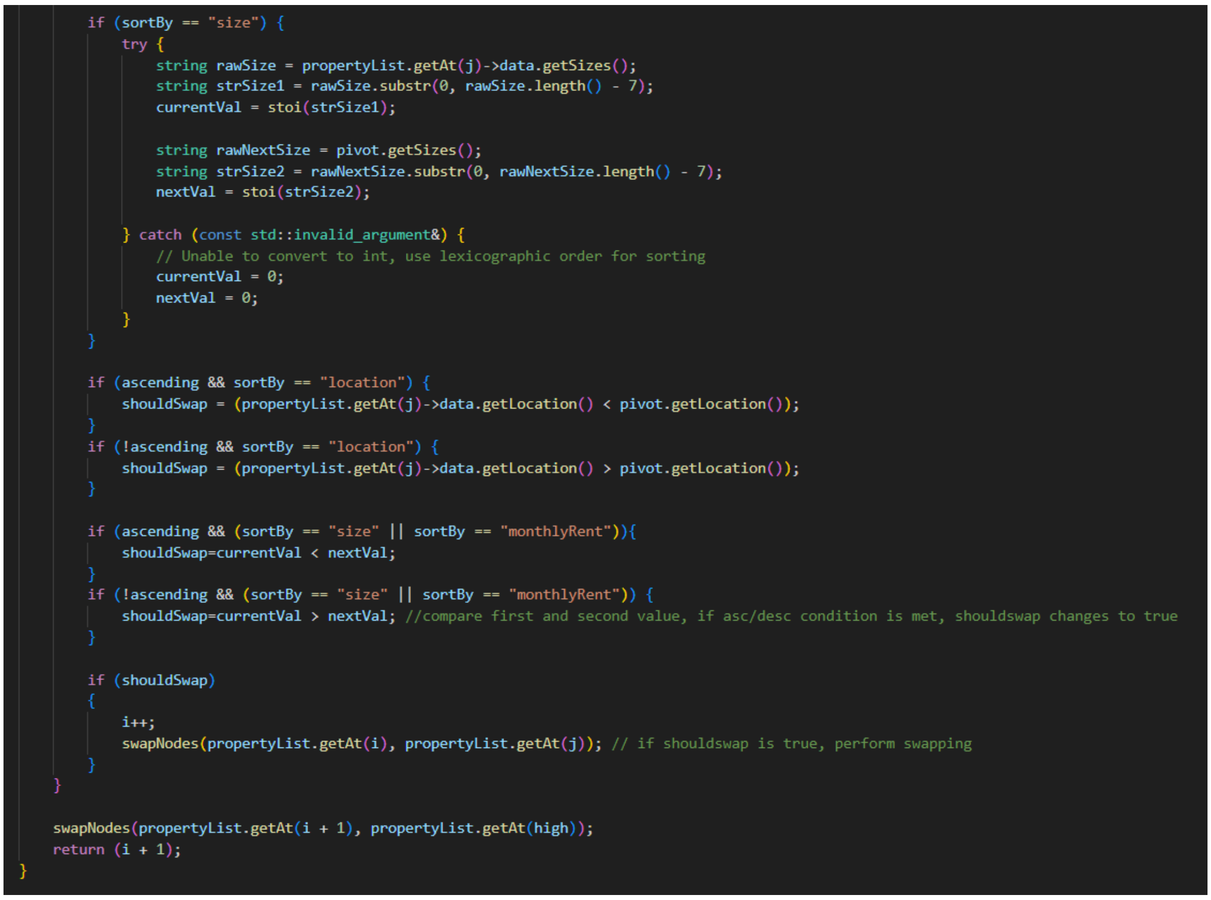
Figure 35 quickSort function

The quickSort function serves a two major functions, which are deciding if the sorting process is complete, where if low is larger or equal to high, it indicates that the entire propertyList have been partitioned and traversed, meaning the list is sorted. Besides that, it also functions to further partition the list into smaller sublists by calling itself based on the "pi" value that is returned from the partition function.



```cpp
template <typename T>
int partition(DoublyLinkedList<T> &propertyList, const string &sortBy, int low, int high, bool ascending) {

    T pivot = propertyList.getAt(high)->data; // Choose the last element as the pivot
    int i = (low - 1); // Index of the smaller element

    for (int j = low; j <= high - 1; j++)
    {
        bool shouldSwap = false;
        int currentVal, nextVal;

        if (sortBy == "monthlyRent") {

            try {
                string rawRent = propertyList.getAt(j)->data.getMonthlyRent();
                if (rawRent.size() >= 12) // checking if the size of recieved string is not out of range
                {
                string strRent1 = rawRent.substr(3, rawRent.length() - 3 - 9);
                strRent1.erase(remove(strRent1.begin(), strRent1.end(), ' '), strRent1.end());
                currentVal = stoi(strRent1); //trim and converting to int
                } else {
                    currentVal = 0; // assign value as 0
                }

                string rawNextRent = pivot.getMonthlyRent();
                if (rawNextRent.size() >= 12) // checking if the size of recieved string is not out of range
                {
                string strRent2 = rawNextRent.substr(3, rawNextRent.length() - 3 - 9);
                strRent2.erase(remove(strRent2.begin(), strRent2.end(), ' '), strRent2.end());
                nextVal = stoi(strRent2); //trim and converting to int
                } else {
                    nextVal = 0; // assign value as 0
                }

            } catch (const std::invalid_argument&) {
                // Unable to convert to int, use lexicographic order for sorting
                currentVal = 0;
                nextVal = 0;
            }
        }
```

Figure 36 partition function (1 of 2)



```
        if (sortBy == "size") {
            try {
                string rawSize = propertyList.getAt(j)->data.getSizes();
                string strSize1 = rawSize.substr(0, rawSize.length() - 7);
                currentVal = stoi(strSize1);

                string rawNextSize = pivot.getSizes();
                string strSize2 = rawNextSize.substr(0, rawNextSize.length() - 7);
                nextVal = stoi(strSize2);

            } catch (const std::invalid_argument&) {
                // Unable to convert to int, use lexicographic order for sorting
                currentVal = 0;
                nextVal = 0;
            }
        }

        if (ascending && sortBy == "location") {
            shouldSwap = (propertyList.getAt(j)->data.getLocation() < pivot.getLocation());
        }
        if (!ascending && sortBy == "location") {
            shouldSwap = (propertyList.getAt(j)->data.getLocation() > pivot.getLocation());
        }

        if (ascending && (sortBy == "size" || sortBy == "monthlyRent")){
            shouldSwap=currentVal < nextVal;
        }
        if (!ascending && (sortBy == "size" || sortBy == "monthlyRent")) {
            shouldSwap=currentVal > nextVal; //compare first and second value, if asc/desc condition is met, shouldswap changes to true
        }

        if (shouldSwap)
        {
            i++;
            swapNodes(propertyList.getAt(i), propertyList.getAt(j)); // if shouldswap is true, perform swapping
        }
    }

    swapNodes(propertyList.getAt(i + 1), propertyList.getAt(high));
    return (i + 1);
}
```

Figure 37 partition function (2 of 2)

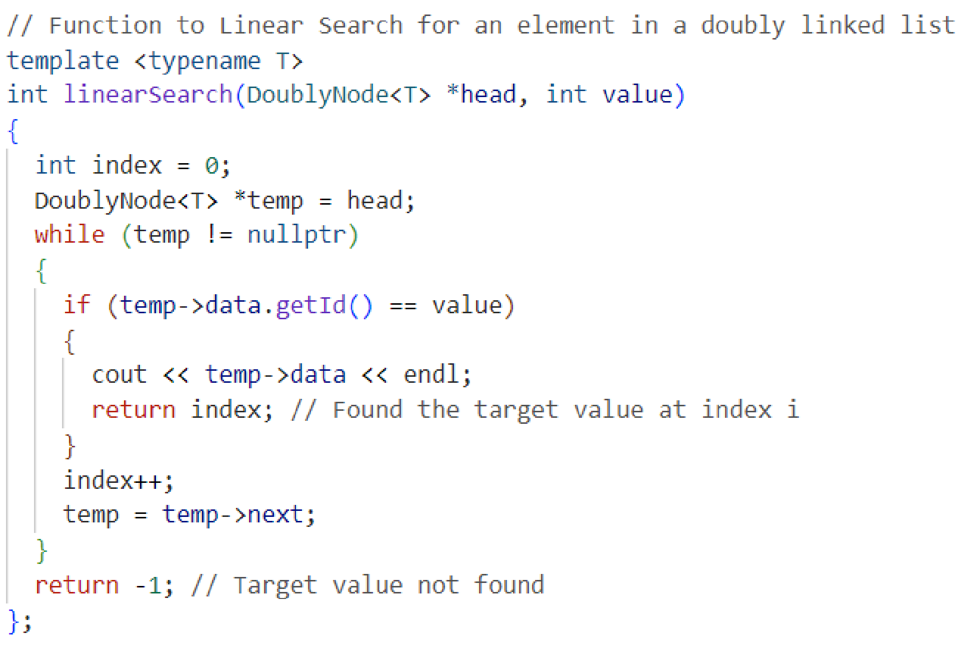
The partition function is responsible for deciding which nodes should be swapped. Similar to bubbleSort, the same data trimming also takes place to convert the data into integer so that it is possible to sort numerically. The function starts by selecting the pivot to be used for comparison by the other data in the sublist. The element “i” refers to an element that is less than or equal to the pivot, while “j” is for traversing through the sublist and checking the value of each element. If shouldSwap is true, that means the element at “j” is smaller than the pivot, and the value of “j” will be swapped with “i”. After this is done, “i” will be incremented by 1, shortening the list of potential swaps. After the for loop completes, the element at “i” is swapped with the pivot, in order to make sure the pivot is in the center and the left is all smaller than the pivot and the right is larger.

With the combination of the partition, quickSort, quickSorts, and swapNodes functions, the proposed Quick Sort algorithm is implemented. As a widely used sorting algorithm, there are many iterations and implementations of the Quick Sort algorithm. One way the implementation was done is in this system, where a pivot is initially selected in the partition function by choosing the last element in the propertyList as the pivot, as well as for every subsequent sublists after each partition. The partition function separates the propertyList into smaller sublists by moving data that is smaller than the pivot into one sublist and bigger data into another sublist. This process is repeated recursively by the quickSort function, where it calls itself to further split into smaller sublists until the entire propertyList is sorted.

2.1.3 Linear Search

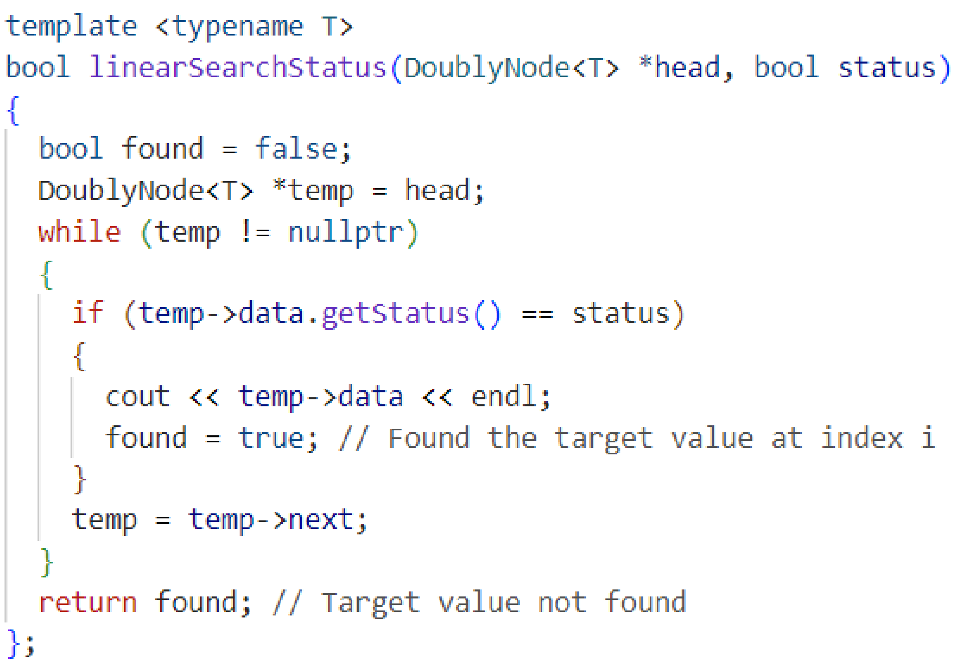
Linear search is a simple and straightforward searching algorithm used to find a target value within a list or array. It involves sequentially checking each element of the list from the beginning to the end until the desired value is found or the entire list has been traversed. If the target value is found, the algorithm returns the index at which it was located; otherwise, it returns an indication that the value is not present in the list.

In the ‘SearchingAlgorithm.h’ header file, There are several templates for linear searches such as ‘linearSearch’ which is a search for a given value in the doubly linked list based on the getId() method of the data object, ‘linearSearchStatus’ which is a search for nodes with a specified status using the getStatus() method, ‘linearSearchGender’ which is a search for nodes with a specified gender using the getGender() method. ‘linearSearchPropertyRoomNumberMoreThan’ which is a search for nodes with room numbers greater than a given value, ‘linearSearchPropertyType’ which is a search for nodes with a specified property type using the getPropertyType() method., ‘linearSearchPropertyRoomNumber’ which is a search for nodes with a specified room number, ‘linearSearchPropertySquareFeetMoreThan’ which is a search for nodes with square footage greater than a given value,, ‘linearSearchPropertySquareFeetLessThan’ which is a search for nodes with square footage less than a given value, ‘linearSearchPropertyPriceMoreThan’ which is a search for nodes with monthly rent greater than a given value, ‘linearSearchPropertyPriceLessThan’ which is a search for nodes with monthly rent less than a given value.



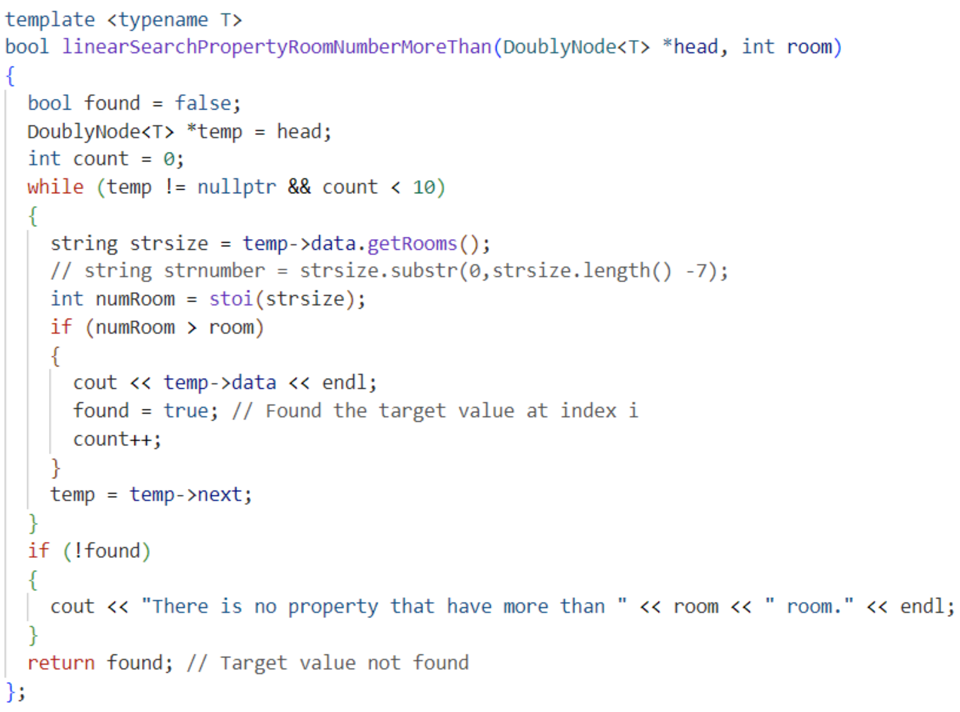
```cpp
// Function to Linear Search for an element in a doubly linked list
template <typename T>
int linearSearch(DoublyNode<T> *head, int value)
{
  int index = 0;
  DoublyNode<T> *temp = head;
  while (temp != nullptr)
  {
    if (temp->data.getId() == value)
    {
      cout << temp->data << endl;
      return index; // Found the target value at index i
    }
    index++;
    temp = temp->next;
  }
  return -1; // Target value not found
};
```

Figure 38: Doubly Linked List Linear Search Template



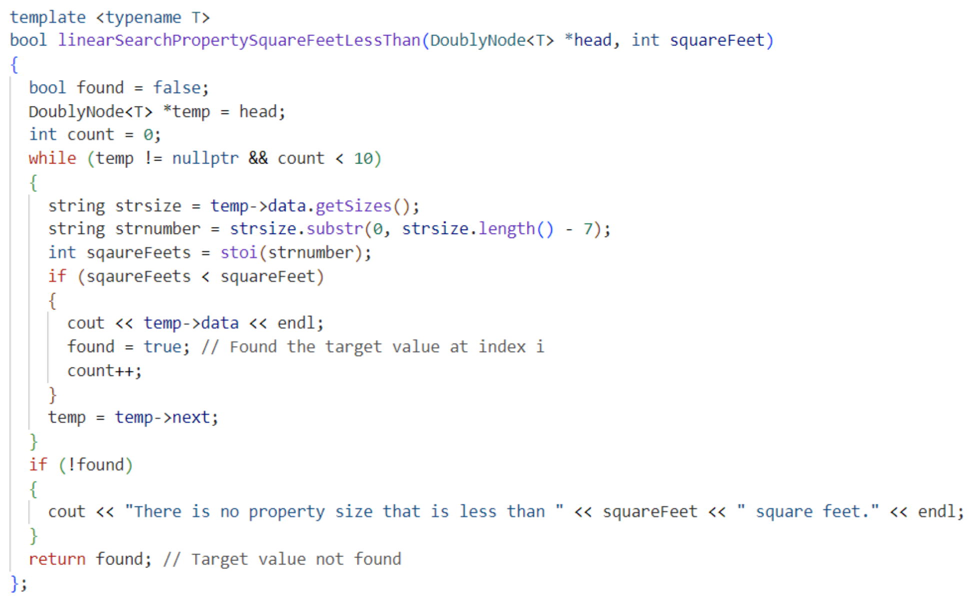
```cpp
template <typename T>
bool linearSearchStatus(DoublyNode<T> *head, bool status)
{
  bool found = false;
  DoublyNode<T> *temp = head;
  while (temp != nullptr)
  {
    if (temp->data.getStatus() == status)
    {
      cout << temp->data << endl;
      found = true; // Found the target value at index i
    }
    temp = temp->next;
  }
  return found; // Target value not found
};
```

Figure 39: Linear Search for Status



```cpp
template <typename T>
bool linearSearchPropertyRoomNumberMoreThan(DoublyNode<T> *head, int room)
{
  bool found = false;
  DoublyNode<T> *temp = head;
  int count = 0;
  while (temp != nullptr && count < 10)
  {
    string strsize = temp->data.getRooms();
    // string strnumber = strsize.substr(0,strsize.length() -7);
    int numRoom = stoi(strsize);
    if (numRoom > room)
    {
      cout << temp->data << endl;
      found = true; // Found the target value at index i
      count++;
    }
    temp = temp->next;
  }
  if (!found)
  {
    cout << "There is no property that have more than " << room << " room." << endl;
  }
  return found; // Target value not found
};
```

Figure 40: Linear Search for Property Room More Than



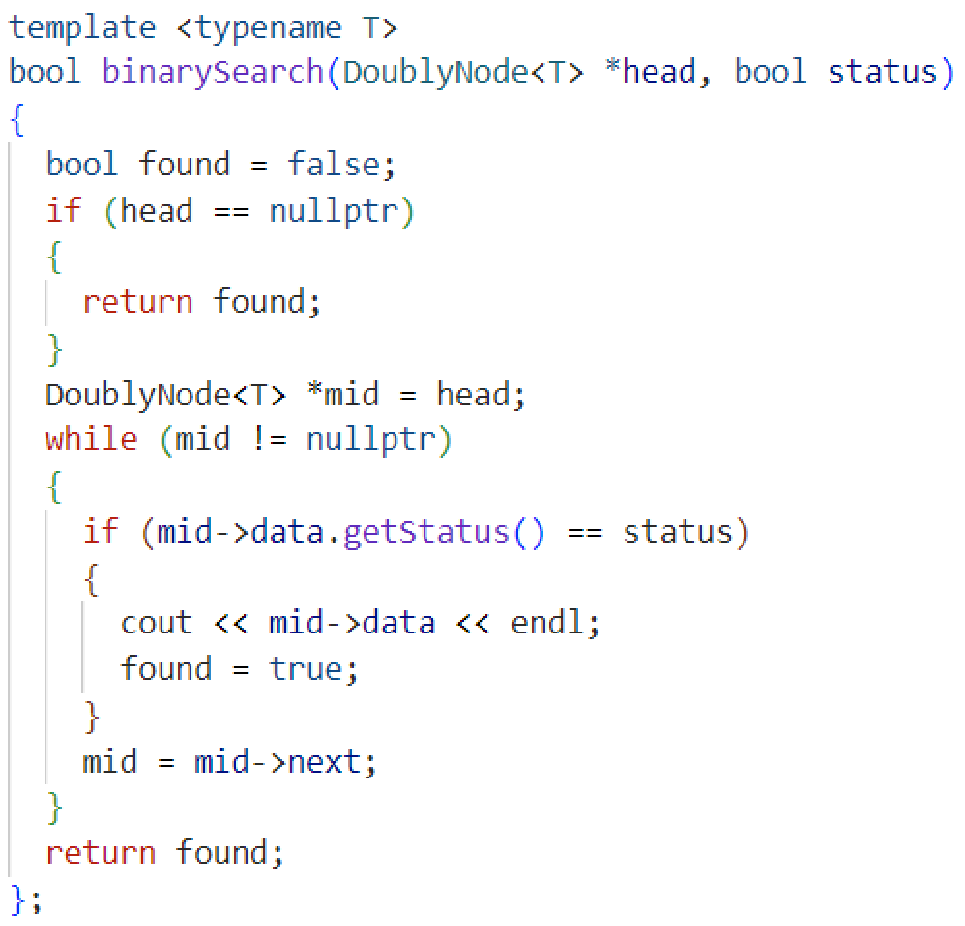
```
template <typename T>
bool linearSearchPropertySquareFeetLessThan(DoublyNode<T> *head, int squareFeet)
{
  bool found = false;
  DoublyNode<T> *temp = head;
  int count = 0;
  while (temp != nullptr && count < 10)
  {
    string strsize = temp->data.getSizes();
    string strnumber = strsize.substr(0, strsize.length() - 7);
    int sqaureFeets = stoi(strnumber);
    if (sqaureFeets < squareFeet)
    {
      cout << temp->data << endl;
      found = true; // Found the target value at index i
      count++;
    }
    temp = temp->next;
  }
  if (!found)
  {
    cout << "There is no property size that is less than " << squareFeet << " square feet." << endl;
  }
  return found; // Target value not found
};
```

2.1.4 Binary Search

Binary search is a more efficient search algorithm that works specifically on sorted lists or arrays. It follows a "divide and conquer" approach to quickly locate a target value by repeatedly halving the search space. It compares the middle element of the current search range with the target value and then narrows down the search to either the left or right half of the range, based on the comparison. This process continues until the target value is found or the search range becomes empty.

Binary search is highly efficient and reduces the search space by half with each iteration. It has a time complexity of O(log n), where n is the number of elements in the list. However, binary search requires the list to be sorted beforehand, and it may not be suitable for unsorted data.

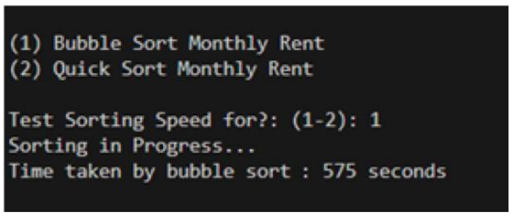
In the ‘SearchingAlgorithm.h’ header file, there is a template function that performs binary search operation on a doubly linked list containing elements of type ‘DoublyNode<T>’. The purpose of the function is to find and print the elements within the doubly linked list whose ‘status’ matches the specified ‘status’ parameter. The function returns ‘true’ if at least one matching element is found; otherwise, it returns ‘false’. In the template function declaration, there are two parameters which are head which is a pointer to the head (start) of the doubly linked list, status which is boolean value indicating the status to search for and the return type of the function is bool, indicating whether any matching elements were found. There is a section that runs initialization and base case check in which that a boolean variable ‘found’ is initialized as ‘false’. This variable will be used to keep track of whether any matching elements were found during the search. If the ‘head’ pointer is ‘nullptr’, which means the linked list is empty, the function immediately returns ‘false’ because there are no elements to search. In search operation, a pointer ‘mid’ is initialized to the ‘head’ of the doubly linked list. A loop iterates through the linked list as long as ‘mid’ is not ‘nullptr’. Within the loop, the ‘getStatus()’ method of the ‘data’ member of the current ‘mid’ node is called to check if the ‘status’ matches the specified ‘status’ parameter. If the status matches, the data of the current node is printed using ‘cout’, and the ‘found’ flag is set to ‘true’ to indicate that a matching element was found. The ‘mid’ pointer is then moved to the next node in the linked list. After iterating through the entire linked list, the function returns the value of the found flag. If ‘found’ is ‘true’, it means at least one matching element was found; otherwise, it returns ‘false’.



```
template <typename T>
bool binarySearch(DoublyNode<T> *head, bool status)
{
    bool found = false;
    if (head == nullptr)
    {
        return found;
    }
    DoublyNode<T> *mid = head;
    while (mid != nullptr)
    {
        if (mid->data.getStatus() == status)
        {
            cout << mid->data << endl;
            found = true;
        }
        mid = mid->next;
    }
    return found;
};
```

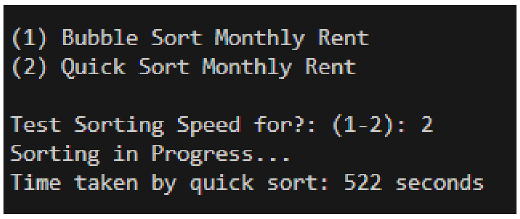
Figure 41: Binary Search Template Function

2.1.5 Comparison Sort-time



```
(1) Bubble Sort Monthly Rent
(2) Quick Sort Monthly Rent

Test Sorting Speed for?: (1-2): 1
Sorting in Progress...
Time taken by bubble sort : 575 seconds
```

Figure 42 Bubble Sort Time Taken



```
(1) Bubble Sort Monthly Rent
(2) Quick Sort Monthly Rent

Test Sorting Speed for?: (1-2): 2
Sorting in Progress...
Time taken by quick sort: 522 seconds
```

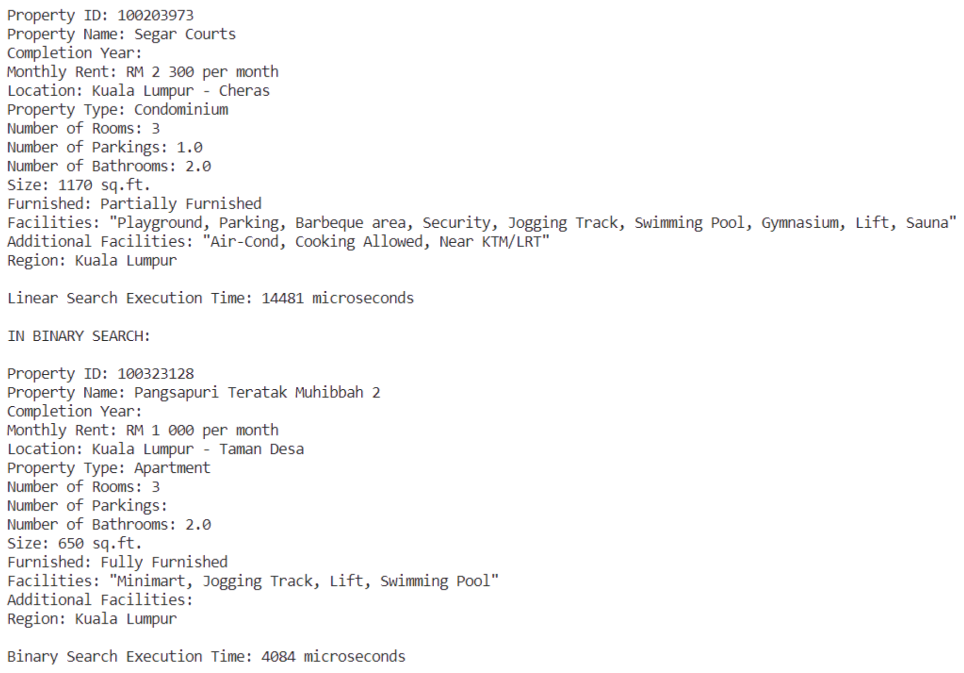
Figure 43 Quick Sort Time Taken

When looking at the time complexity, it is observed that quick sort is generally faster than bubble sort, as the time complexity for quick sort is O(n log n) and the time complexity for bubble sort is O(n^2). After testing both algorithms using the data structures as well as the provided csv file, it is observed that quick sort is faster than bubble sort by a little more than a minute.

It is also noted that bubble sort works better for smaller datasets while quicksort is preferred for larger datasets. In the case of the Asia Pacific Home System, having to sort through the csv file which contains 19991 records of property data, the quick sort algorithm came out on top against bubble sort.

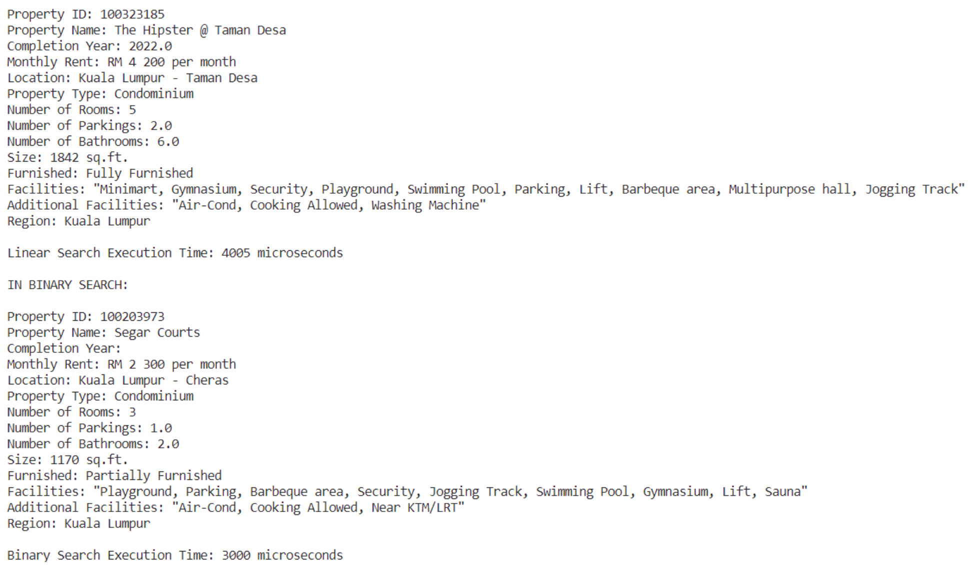
The execution time of both algorithms may be further shortened however, as certain optimizations to the code might be possible to help reduce the steps needed to be executed in order to achieve the same results. For example, in the bubbleSort function, if the largest value is finished sorting for the first loop, that value can be ignored for the subsequent loops so that it takes less steps for comparing values before entering the next loop. Other than that, both sorting algorithms in the system can also benefit from having the data conversion for monthly rent and size from string to integer to be done beforehand before entering the sorting functions. This also reduces the tasks needed to be done before actually comparing and sorting the data.

2.1.6 Comparison Search Time



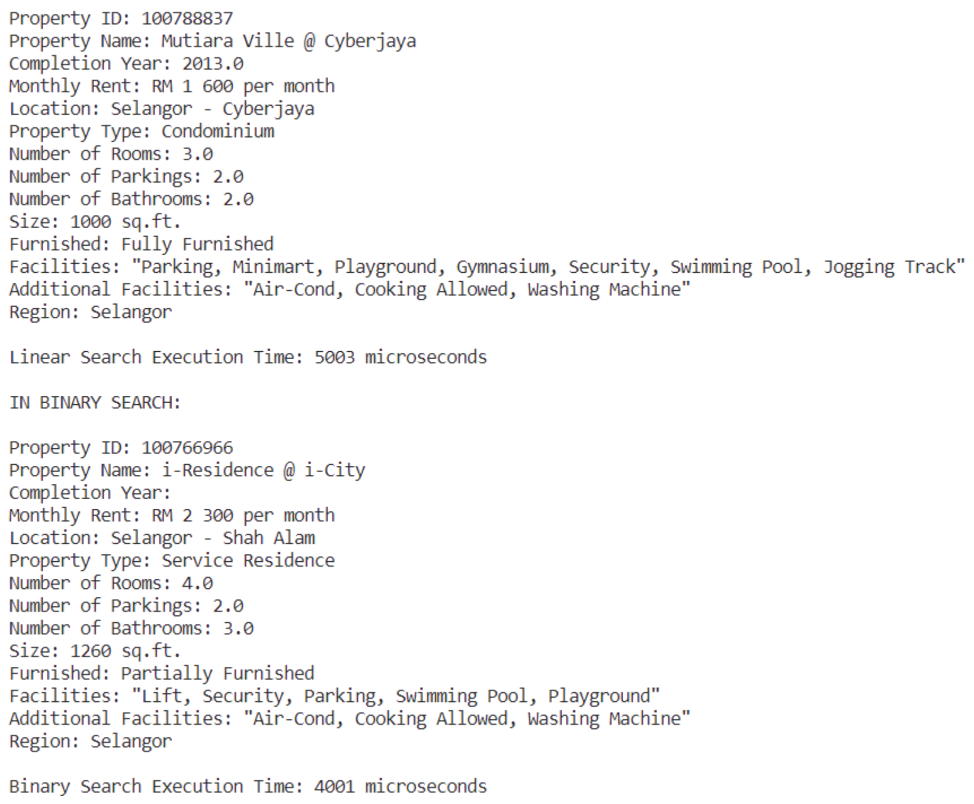
```
Property ID: 100203973
Property Name: Segar Courts
Completion Year:
Monthly Rent: RM 2 300 per month
Location: Kuala Lumpur - Cheras
Property Type: Condominium
Number of Rooms: 3
Number of Parkings: 1.0
Number of Bathrooms: 2.0
Size: 1170 sq.ft.
Furnished: Partially Furnished
Facilities: "Playground, Parking, Barbeque area, Security, Jogging Track, Swimming Pool, Gymnasium, Lift, Sauna"
Additional Facilities: "Air-Cond, Cooking Allowed, Near KTM/LRT"
Region: Kuala Lumpur

Linear Search Execution Time: 14481 microseconds

IN BINARY SEARCH:

Property ID: 100323128
Property Name: Pangsapuri Teratak Muhibbah 2
Completion Year:
Monthly Rent: RM 1 000 per month
Location: Kuala Lumpur - Taman Desa
Property Type: Apartment
Number of Rooms: 3
Number of Parkings:
Number of Bathrooms: 2.0
Size: 650 sq.ft.
Furnished: Fully Furnished
Facilities: "Minimart, Jogging Track, Lift, Swimming Pool"
Additional Facilities:
Region: Kuala Lumpur

Binary Search Execution Time: 4084 microseconds
```

Figure 44: Linear Search and Binary Search Execution Time for Property Room



```
Property ID: 100323185
Property Name: The Hipster @ Taman Desa
Completion Year: 2022.0
Monthly Rent: RM 4 200 per month
Location: Kuala Lumpur - Taman Desa
Property Type: Condominium
Number of Rooms: 5
Number of Parkings: 2.0
Number of Bathrooms: 6.0
Size: 1842 sq.ft.
Furnished: Fully Furnished
Facilities: "Minimart, Gymnasium, Security, Playground, Swimming Pool, Parking, Lift, Barbeque area, Multipurpose hall, Jogging Track"
Additional Facilities: "Air-Cond, Cooking Allowed, Washing Machine"
Region: Kuala Lumpur

Linear Search Execution Time: 4005 microseconds

IN BINARY SEARCH:

Property ID: 100203973
Property Name: Segar Courts
Completion Year:
Monthly Rent: RM 2 300 per month
Location: Kuala Lumpur - Cheras
Property Type: Condominium
Number of Rooms: 3
Number of Parkings: 1.0
Number of Bathrooms: 2.0
Size: 1170 sq.ft.
Furnished: Partially Furnished
Facilities: "Playground, Parking, Barbeque area, Security, Jogging Track, Swimming Pool, Gymnasium, Lift, Sauna"
Additional Facilities: "Air-Cond, Cooking Allowed, Near KTM/LRT"
Region: Kuala Lumpur

Binary Search Execution Time: 3000 microseconds
```

Figure 45: Linear Search and Binary Search Execution Time for Property Type



```
Property ID: 100788837
Property Name: Mutiara Ville @ Cyberjaya
Completion Year: 2013.0
Monthly Rent: RM 1 600 per month
Location: Selangor - Cyberjaya
Property Type: Condominium
Number of Rooms: 3.0
Number of Parkings: 2.0
Number of Bathrooms: 2.0
Size: 1000 sq.ft.
Furnished: Fully Furnished
Facilities: "Parking, Minimart, Playground, Gymnasium, Security, Swimming Pool, Jogging Track"
Additional Facilities: "Air-Cond, Cooking Allowed, Washing Machine"
Region: Selangor

Linear Search Execution Time: 5003 microseconds

IN BINARY SEARCH:

Property ID: 100766966
Property Name: i-Residence @ i-City
Completion Year:
Monthly Rent: RM 2 300 per month
Location: Selangor - Shah Alam
Property Type: Service Residence
Number of Rooms: 4.0
Number of Parkings: 2.0
Number of Bathrooms: 3.0
Size: 1260 sq.ft.
Furnished: Partially Furnished
Facilities: "Lift, Security, Parking, Swimming Pool, Playground"
Additional Facilities: "Air-Cond, Cooking Allowed, Washing Machine"
Region: Selangor

Binary Search Execution Time: 4001 microseconds
```

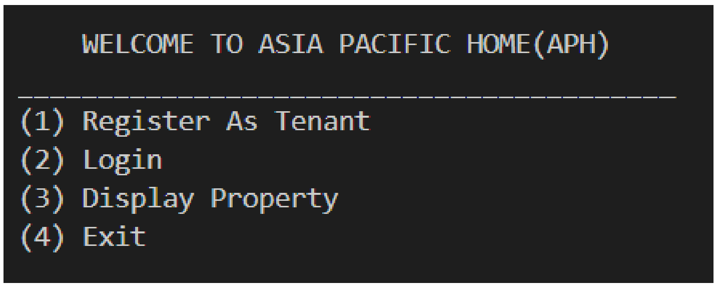
Figure 46: Linear Search and Binary Search Execution Time for Property Region

Based on the figures, it seems that after running on different search testes, execution time of binary search is always shorter than execution time of linear search. It is because there are some major differences between them in how they narrow down the search space and locate the desired element. In a linear search, you start at one end of the list and compare each element sequentially until you find the target or reach the end of the list. This means you might need to go through every element in the worst case, leading to a time complexity of O(n), where n is the number of elements in the list. Binary search works by repeatedly dividing the search space in half. You compare the middle element of the current search space to the target. If the target is less than the middle element, you focus on the left half; if it's greater, you focus on the right half. This process significantly reduces the search space with each step, resulting in a time complexity of O(log n) in the worst case. In a linear search, the number of comparisons is directly proportional to the number of elements in the list. The worst case requires checking all elements. With each comparison in binary search, the search space is halved. This means that for a list of n elements, after the first comparison, you're left with at most n/2 elements, then n/4, n/8, and so on. This logarithmic reduction in search space leads to far fewer comparisons in the worst case compared to linear search. In linear search, O(n) time complexity means that the execution time grows linearly with the size of the input. In binary search, O(log n) time complexity indicates that the execution time grows logarithmically with the size of the input. Logarithmic growth is much slower than linear growth, especially for large inputs.

3.0 System Input /Output

3.1 All user

3.1.1 Main Menu

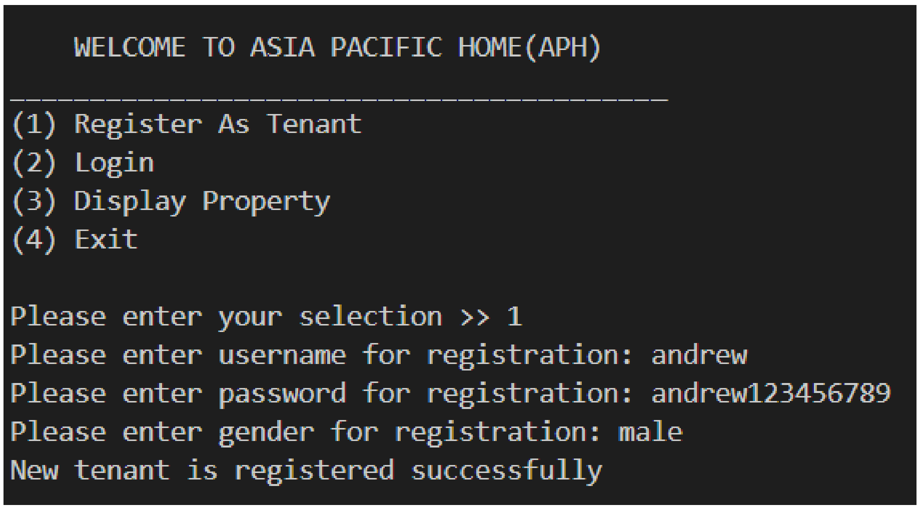


```
    WELCOME TO ASIA PACIFIC HOME(APH)

__________________________________________
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit
```

Figure 47: APH Main Menu

Diagram above are the Asia Pacific Home main menu. User are allow to register as a new tenant, login based on their roles and credential, display the property , and click exit to exit this program.

3.1.2 Register as tenant



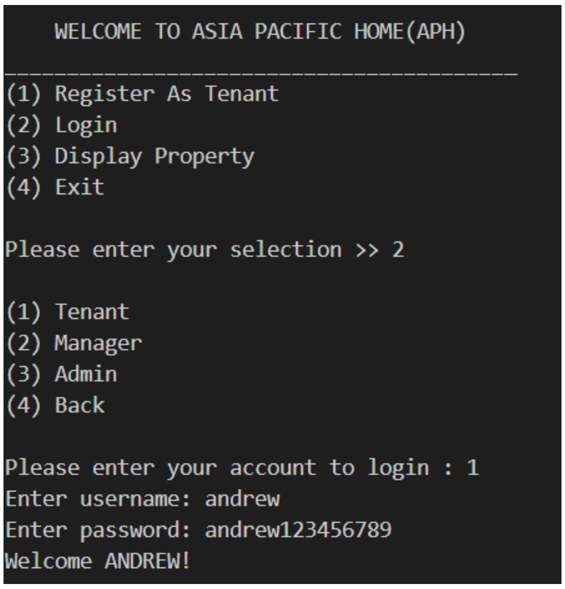
```
    WELCOME TO ASIA PACIFIC HOME(APH)

__________________________________________
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit

Please enter your selection >> 1
Please enter username for registration: andrew
Please enter password for registration: andrew123456789
Please enter gender for registration: male
New tenant is registered successfully
```

Figure 48: Register as tenant function

Diagram above show the register as tenant function which allow user to register as a new tenant . First, user will require to enter their username and password for registration and enter their gender as well. our system will set the tenant status to ACTIVE user as default.

3.1.3 Login

This is the login function for user to select their role and enter their credential as well. the

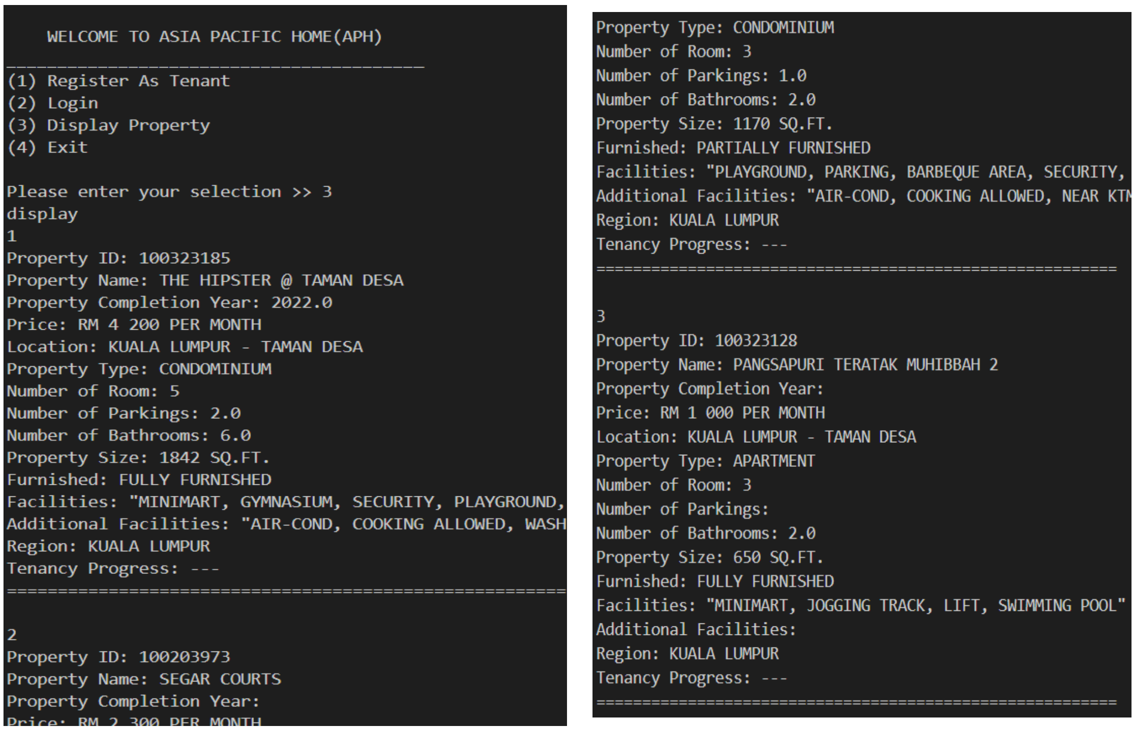


```
   WELCOME TO ASIA PACIFIC HOME(APH)

-----------------------------------------
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit

Please enter your selection >> 2

(1) Tenant
(2) Manager
(3) Admin
(4) Back

Please enter your account to login : 1
Enter username: andrew
Enter password: andrew123456789
Welcome ANDREW!
```

Figure 49: Login function

APH system has 3 user role which is tenant , manager, and admin. Once user choose their role, they require to enter the credential. If the user credential is exist in the user list , then a message will display to welcome the user .

3.1.4 Display property



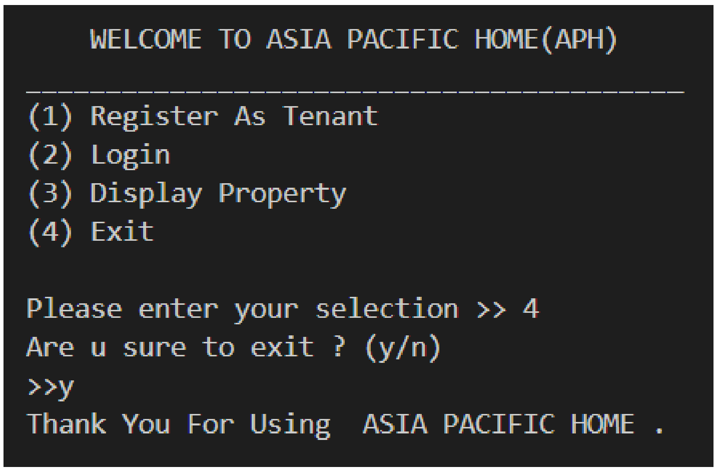
```
   WELCOME TO ASIA PACIFIC HOME(APH)

-----------------------------------------
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit

Please enter your selection >> 3
display
1
Property ID: 100323185
Property Name: THE HIPSTER @ TAMAN DESA
Property Completion Year: 2022.0
Price: RM 4 200 PER MONTH
Location: KUALA LUMPUR - TAMAN DESA
Property Type: CONDOMINIUM
Number of Room: 5
Number of Parkings: 2.0
Number of Bathrooms: 6.0
Property Size: 1842 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: "MINIMART, GYMNASIUM, SECURITY, PLAYGROUND,
Additional Facilities: "AIR-COND, COOKING ALLOWED, WASH
Region: KUALA LUMPUR
Tenancy Progress: ---
=======================================================

2
Property ID: 100203973
Property Name: SEGAR COURTS
Property Completion Year:
Price: RM 2 300 PER MONTH
```

```
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1.0
Number of Bathrooms: 2.0
Property Size: 1170 SQ.FT.
Furnished: PARTIALLY FURNISHED
Facilities: "PLAYGROUND, PARKING, BARBEQUE AREA, SECURITY,
Additional Facilities: "AIR-COND, COOKING ALLOWED, NEAR KTM
Region: KUALA LUMPUR
Tenancy Progress: ---
=======================================================

3
Property ID: 100323128
Property Name: PANGSAPURI TERATAK MUHIBBAH 2
Property Completion Year:
Price: RM 1 000 PER MONTH
Location: KUALA LUMPUR - TAMAN DESA
Property Type: APARTMENT
Number of Room: 3
Number of Parkings:
Number of Bathrooms: 2.0
Property Size: 650 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: "MINIMART, JOGGING TRACK, LIFT, SWIMMING POOL"
Additional Facilities:
Region: KUALA LUMPUR
Tenancy Progress: ---
=======================================================
```

Figure 50: Display property function

This is the display property function to display the property information in APH system.

3.1.5 Exit function

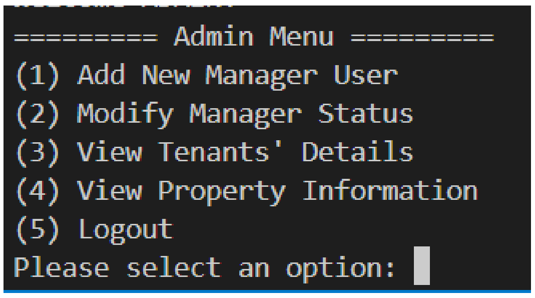


```
    WELCOME TO ASIA PACIFIC HOME(APH)
_________________________________________
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit

Please enter your selection >> 4
Are u sure to exit ? (y/n)
>>y
Thank You For Using  ASIA PACIFIC HOME .
```

Figure 51: Exit function

User can use this exit function to terminate and exit the APH system. When user click to exit, the system will display a message to ask for user confirmation . if user click “y” then the system will terminate .

3.2 Admin

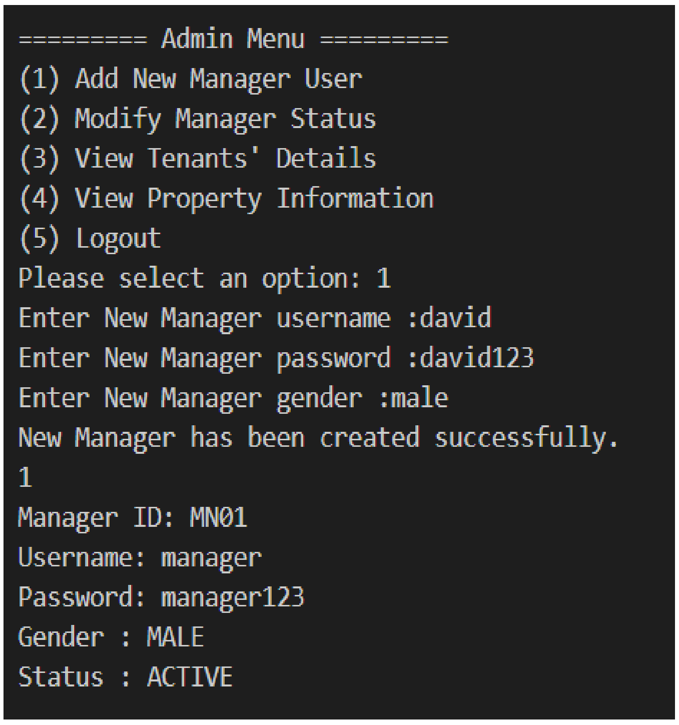


```
========= Admin Menu =========
(1) Add New Manager User
(2) Modify Manager Status
(3) View Tenants' Details
(4) View Property Information
(5) Logout
Please select an option:
```

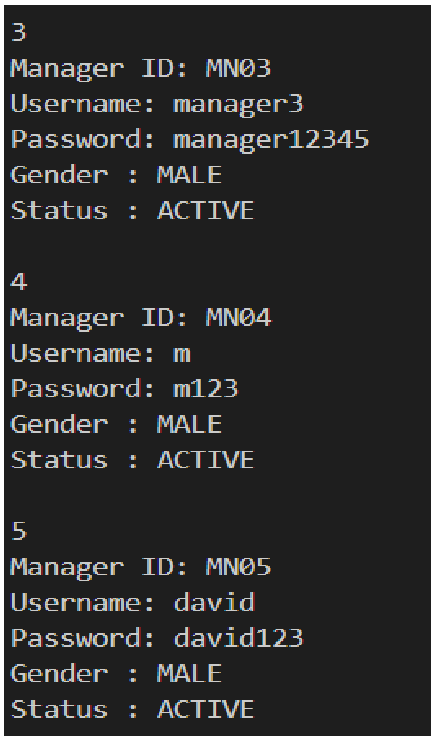
Figure 52: Admin Menu

Diagram above are the admin menu that allow admin to add new manager, modify manager status ,view tenant details ,view property information and logout.

3.2 1 Add new manager



```
========= Admin Menu =========
(1) Add New Manager User
(2) Modify Manager Status
(3) View Tenants' Details
(4) View Property Information
(5) Logout
Please select an option: 1
Enter New Manager username :david
Enter New Manager password :david123
Enter New Manager gender :male
New Manager has been created successfully.
1
Manager ID: MN01
Username: manager
Password: manager123
Gender : MALE
Status : ACTIVE
```

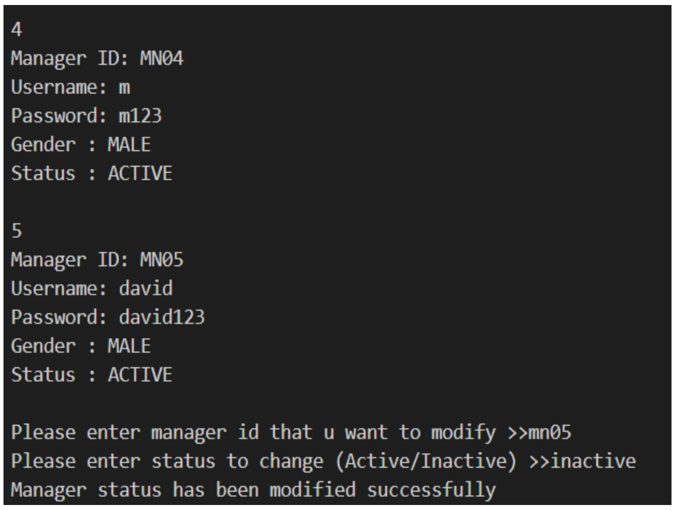


```
3
Manager ID: MN03
Username: manager3
Password: manager12345
Gender : MALE
Status : ACTIVE

4
Manager ID: MN04
Username: m
Password: m123
Gender : MALE
Status : ACTIVE

5
Manager ID: MN05
Username: david
Password: david123
Gender : MALE
Status : ACTIVE
```

Figure 53: Add new manager function

Diagram above show the add new manager function to allow admin to add new manager, admin will need to enter the manager credential, gender . The manager status will automatically set to ACTIVE as default. Once the manager has been created successfully , the system will show all the manager list .

3.2.2 Modify manager status


```
4
Manager ID: MN04
Username: m
Password: m123
Gender : MALE
Status : ACTIVE

5
Manager ID: MN05
Username: david
Password: david123
Gender : MALE
Status : ACTIVE

Please enter manager id that u want to modify >>mn05
Please enter status to change (Active/Inactive) >>inactive
Manager status has been modified successfully
```

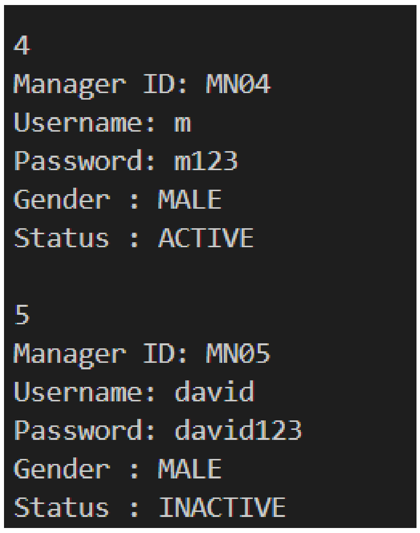

```
4
Manager ID: MN04
Username: m
Password: m123
Gender : MALE
Status : ACTIVE

5
Manager ID: MN05
Username: david
Password: david123
Gender : MALE
Status : INACTIVE
```

Figure 54: Modify manager status function

When admin click the modify manager status button, the system will first display all the manager list information the ask admin for the manager id that want to modify and status to change. Once the manager status has been modified , the system will then display the latest manager information .

3.2.3 View tenant detail

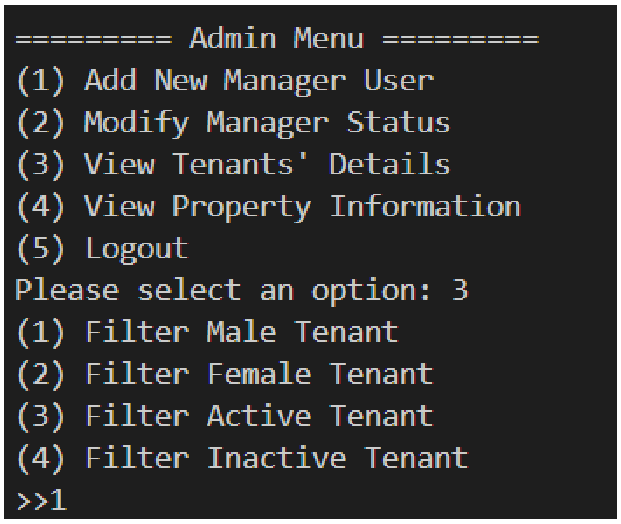
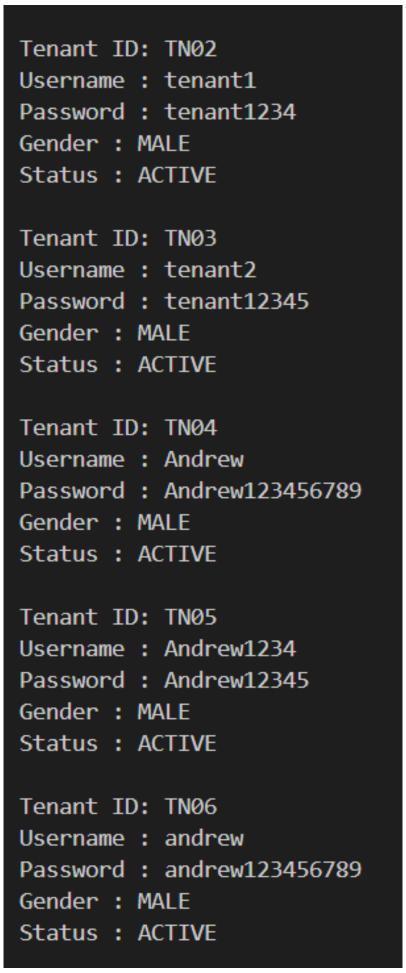
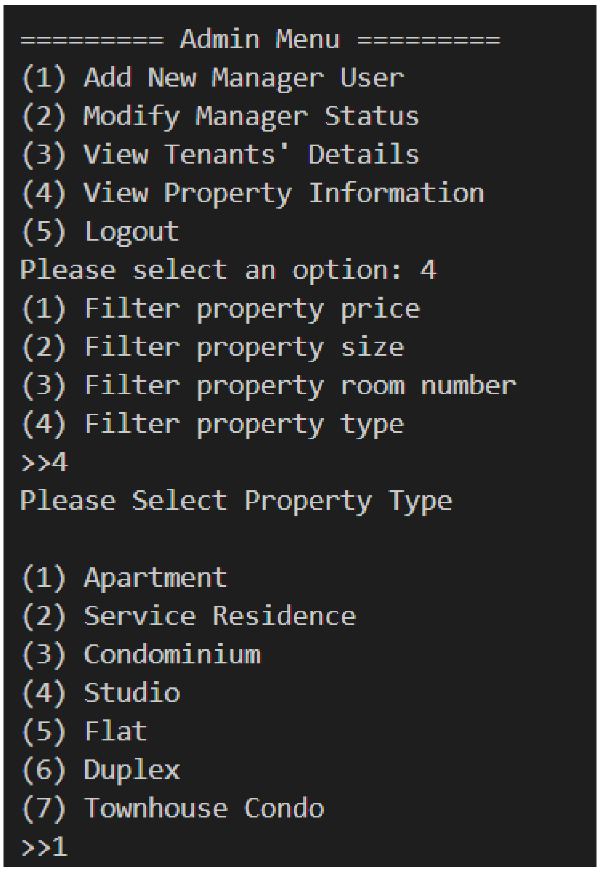

```
========= Admin Menu =========
(1) Add New Manager User
(2) Modify Manager Status
(3) View Tenants' Details
(4) View Property Information
(5) Logout
Please select an option: 3
(1) Filter Male Tenant
(2) Filter Female Tenant
(3) Filter Active Tenant
(4) Filter Inactive Tenant
>>1
```

Figure 55: view tenant detail

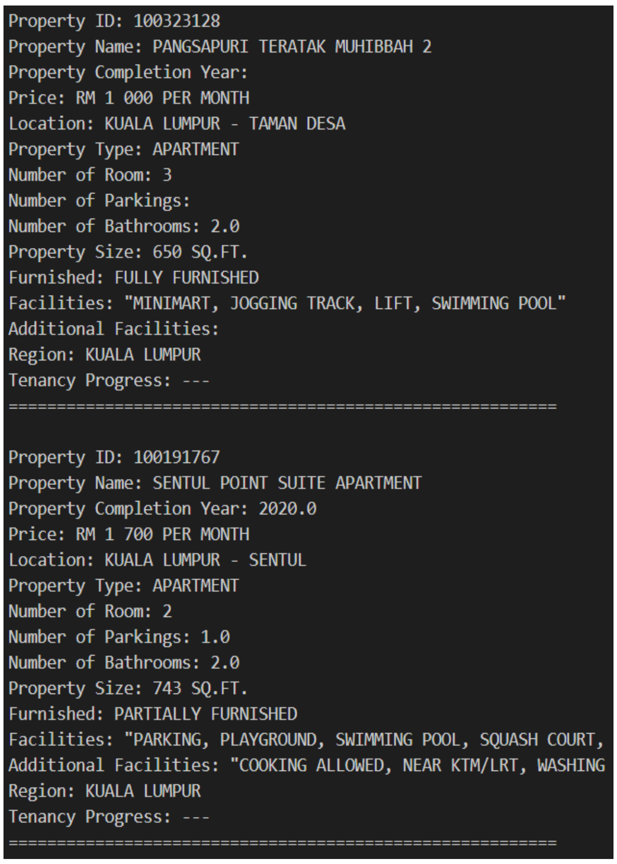

```
Tenant ID: TN02
Username : tenant1
Password : tenant1234
Gender : MALE
Status : ACTIVE

Tenant ID: TN03
Username : tenant2
Password : tenant12345
Gender : MALE
Status : ACTIVE

Tenant ID: TN04
Username : Andrew
Password : Andrew123456789
Gender : MALE
Status : ACTIVE

Tenant ID: TN05
Username : Andrew1234
Password : Andrew12345
Gender : MALE
Status : ACTIVE

Tenant ID: TN06
Username : andrew
Password : andrew123456789
Gender : MALE
Status : ACTIVE
```

Next, diagram above and diagram right show the view tenant details function. Admin are allow to filter and view the tenant detail. In this case, we have take filter male tenant as an example. We can see that the tenant list that show on the right side are all male tenant user. Beside than filtering by tenant's gender , admin are also allow to filter the tenant based on their status.

3.2.4 View property information



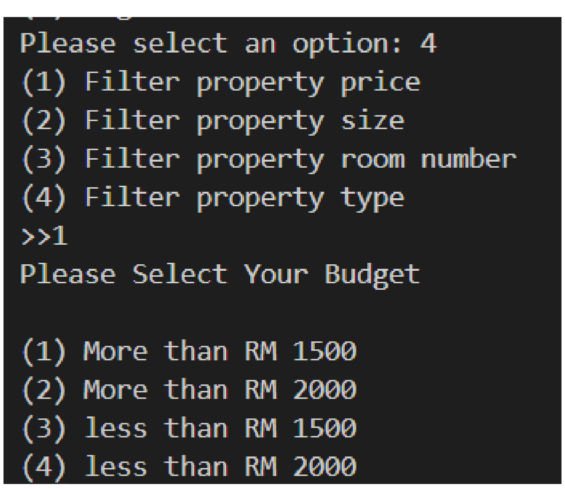
```
========= Admin Menu =========
(1) Add New Manager User
(2) Modify Manager Status
(3) View Tenants' Details
(4) View Property Information
(5) Logout
Please select an option: 4
(1) Filter property price
(2) Filter property size
(3) Filter property room number
(4) Filter property type
>>4
Please Select Property Type

(1) Apartment
(2) Service Residence
(3) Condominium
(4) Studio
(5) Flat
(6) Duplex
(7) Townhouse Condo
>>1
```



```
Property ID: 100323128
Property Name: PANGSAPURI TERATAK MUHIBBAH 2
Property Completion Year:
Price: RM 1 000 PER MONTH
Location: KUALA LUMPUR - TAMAN DESA
Property Type: APARTMENT
Number of Room: 3
Number of Parkings:
Number of Bathrooms: 2.0
Property Size: 650 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: "MINIMART, JOGGING TRACK, LIFT, SWIMMING POOL"
Additional Facilities:
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================

Property ID: 100191767
Property Name: SENTUL POINT SUITE APARTMENT
Property Completion Year: 2020.0
Price: RM 1 700 PER MONTH
Location: KUALA LUMPUR - SENTUL
Property Type: APARTMENT
Number of Room: 2
Number of Parkings: 1.0
Number of Bathrooms: 2.0
Property Size: 743 SQ.FT.
Furnished: PARTIALLY FURNISHED
Facilities: "PARKING, PLAYGROUND, SWIMMING POOL, SQUASH COURT,
Additional Facilities: "COOKING ALLOWED, NEAR KTM/LRT, WASHING
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================
```

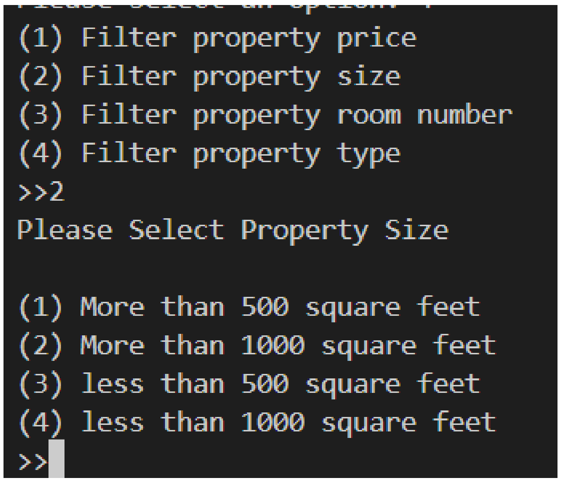
Figure 56: View property information

Diagram above is the view property information, admin can filter and view the property information by filtering their property price, property size, property room number and property type as well. In this case, we have choose filter property type as example. We has select to filter the property by property type apartment only. Hence , the property list that only show apartment will then be show in the terminal.



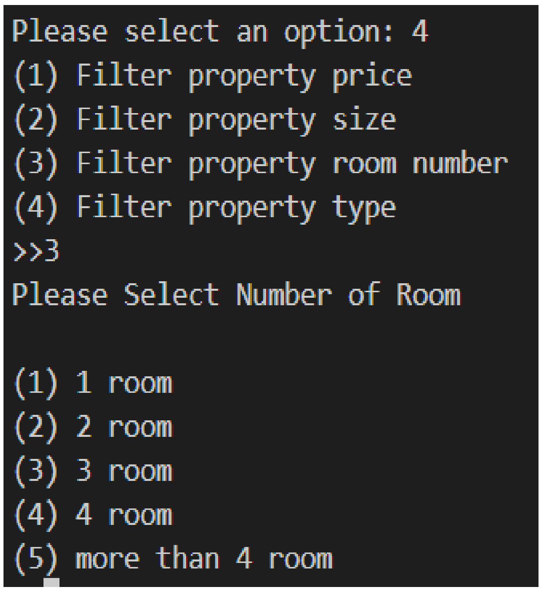
```
Please select an option: 4
(1) Filter property price
(2) Filter property size
(3) Filter property room number
(4) Filter property type
>>1
Please Select Your Budget

(1) More than RM 1500
(2) More than RM 2000
(3) less than RM 1500
(4) less than RM 2000
```

Figure 58: Filter property price



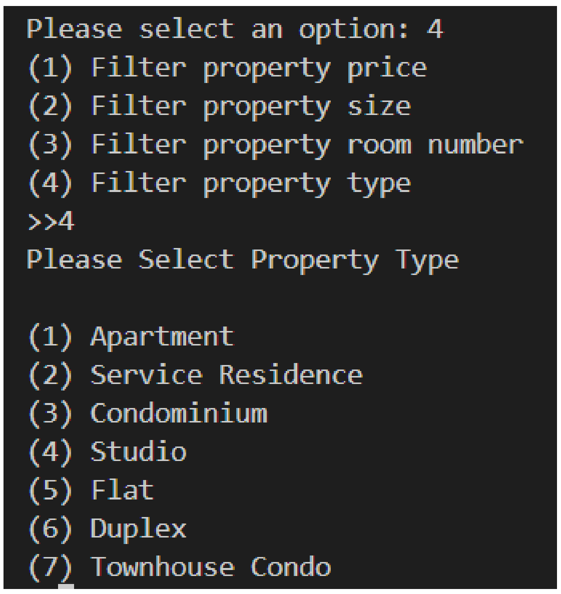
```
(1) Filter property price
(2) Filter property size
(3) Filter property room number
(4) Filter property type
>>2
Please Select Property Size

(1) More than 500 square feet
(2) More than 1000 square feet
(3) less than 500 square feet
(4) less than 1000 square feet
>>
```

Figure 57: Filter property size



```
Please select an option: 4
(1) Filter property price
(2) Filter property size
(3) Filter property room number
(4) Filter property type
>>3
Please Select Number of Room

(1) 1 room
(2) 2 room
(3) 3 room
(4) 4 room
(5) more than 4 room
```

Figure 60: Filter property room number

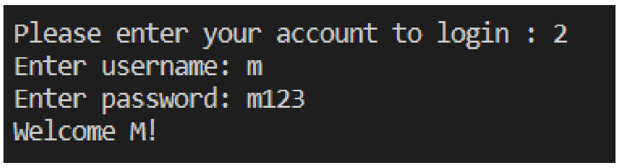


```
Please select an option: 4
(1) Filter property price
(2) Filter property size
(3) Filter property room number
(4) Filter property type
>>4
Please Select Property Type

(1) Apartment
(2) Service Residence
(3) Condominium
(4) Studio
(5) Flat
(6) Duplex
(7) Townhouse Condo
```

Figure 59: Filter Property type

3.3 Managers

3.3.1 Login

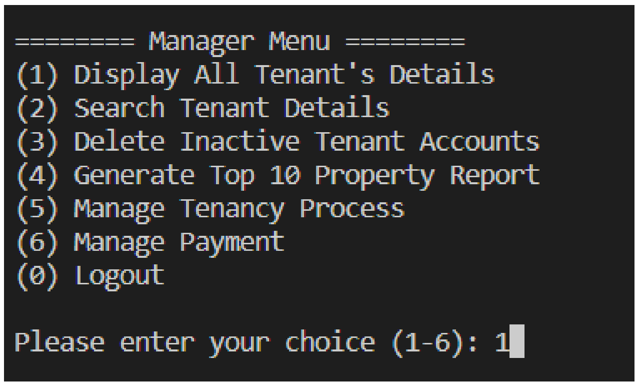


```
Please enter your account to login : 2
Enter username: m
Enter password: m123
Welcome M!
```

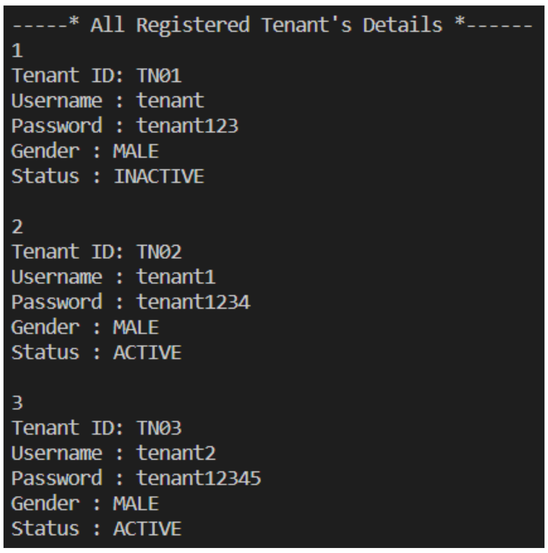
Figure 61 Manager login function

The figure above indicates the manager login section. After the user input 2 for the role account to login, the system will pass it to the manager login path. If the username and password are correct, it represents login successfully, and a welcome message will print out.

3.3.2 Display all Registered Tenant's Details



```
======== Manager Menu ========
(1) Display All Tenant's Details
(2) Search Tenant Details
(3) Delete Inactive Tenant Accounts
(4) Generate Top 10 Property Report
(5) Manage Tenancy Process
(6) Manage Payment
(0) Logout

Please enter your choice (1-6): 1
```

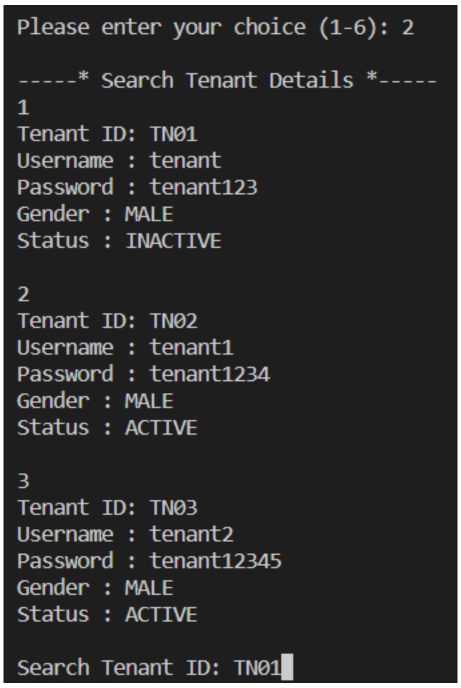


```
-----* All Registered Tenant's Details *------
1
Tenant ID: TN01
Username : tenant
Password : tenant123
Gender : MALE
Status : INACTIVE

2
Tenant ID: TN02
Username : tenant1
Password : tenant1234
Gender : MALE
Status : ACTIVE

3
Tenant ID: TN03
Username : tenant2
Password : tenant12345
Gender : MALE
Status : ACTIVE
```

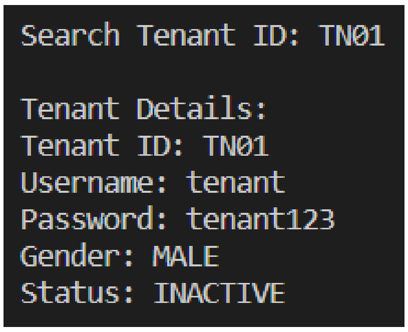
Figure 62 Manager Main menu & display all tenant's Details function

From the manager menu list, there is 7 function to execute, which the first is to display all tenant's details. When the user input 1, the function is executed and will show all the tenant detail in a list. The details will include tenant ID, username, password, gender, and status. As in the figure shown above, three registered tenant details are shown.

3.3.3 Search Tenant's Details



```
Please enter your choice (1-6): 2

-----* Search Tenant Details *-----
1
Tenant ID: TN01
Username : tenant
Password : tenant123
Gender : MALE
Status : INACTIVE

2
Tenant ID: TN02
Username : tenant1
Password : tenant1234
Gender : MALE
Status : ACTIVE

3
Tenant ID: TN03
Username : tenant2
Password : tenant12345
Gender : MALE
Status : ACTIVE

Search Tenant ID: TN01
```

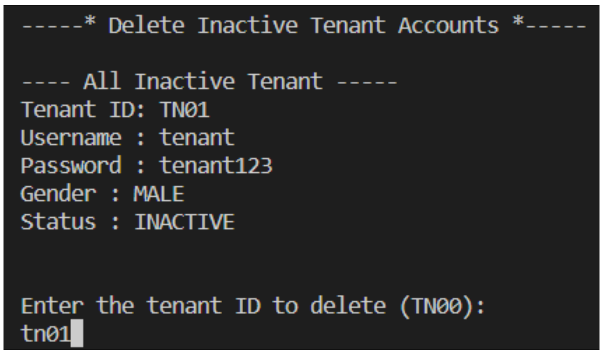


```
Search Tenant ID: TN01

Tenant Details:
Tenant ID: TN01
Username: tenant
Password: tenant123
Gender: MALE
Status: INACTIVE
```

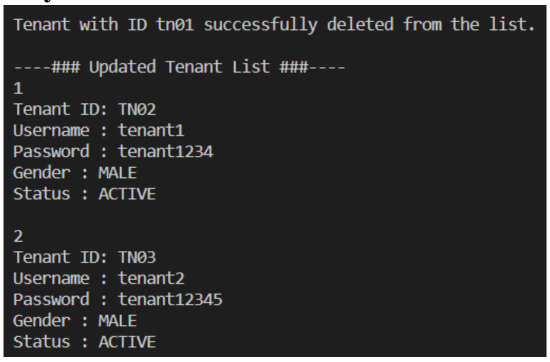
Figure 63 Search tenant detail function

The system will navigate back to the manager's main menu by completing one function. The user must input 2 in the manager main menu part to access the second function. By showing all the tenant details, a tenant ID input will be needed for the user to enter which tenant will search. For example, when entering TN01, the system will return the TN01 tenant's detail.

3.3.4 Delete Tenant Accounts based on Inactivity Status.



```
-----* Delete Inactive Tenant Accounts *-----

---- All Inactive Tenant -----
Tenant ID: TN01
Username : tenant
Password : tenant123
Gender : MALE
Status : INACTIVE


Enter the tenant ID to delete (TN00):
tn01
```

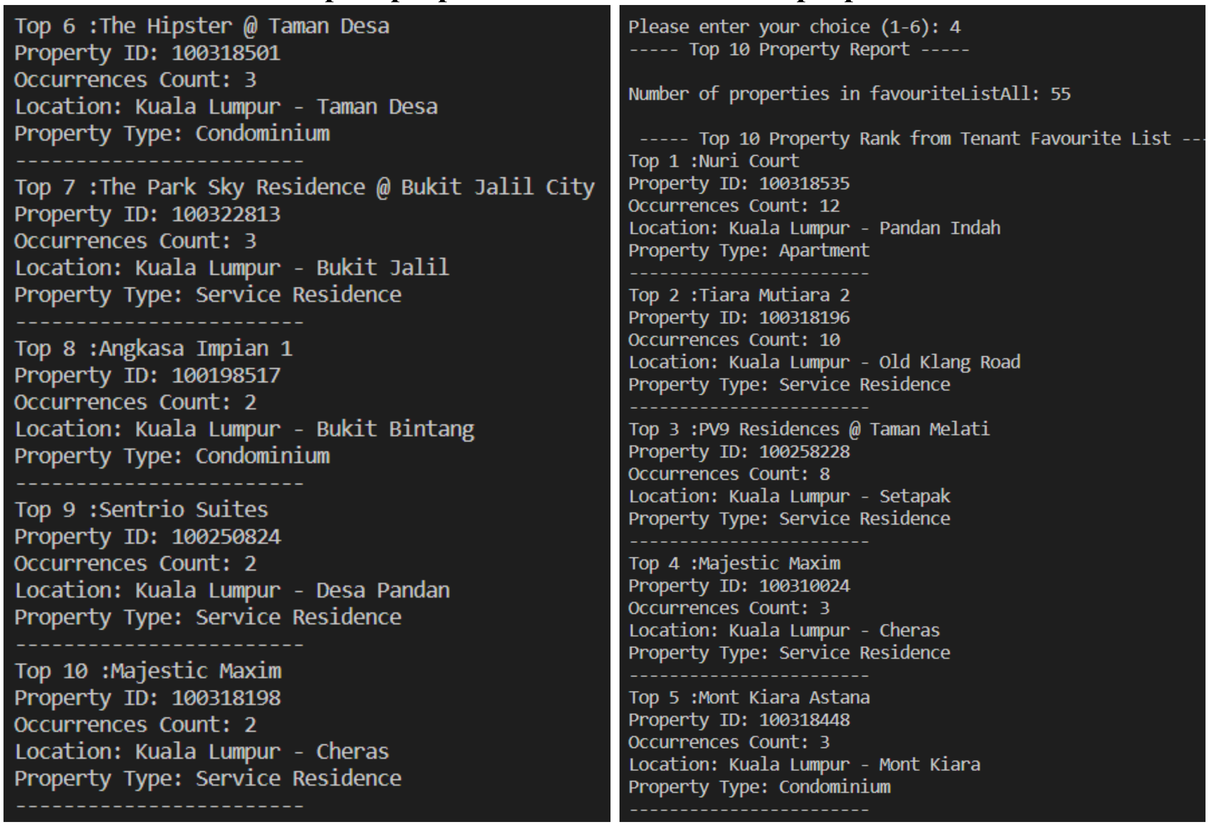


```
Tenant with ID tn01 successfully deleted from the list.

----### Updated Tenant List ###----
1
Tenant ID: TN02
Username : tenant1
Password : tenant1234
Gender : MALE
Status : ACTIVE

2
Tenant ID: TN03
Username : tenant2
Password : tenant12345
Gender : MALE
Status : ACTIVE
```

Figure 64 Delete tenant account based on inactivity status function.

The third function must enter the tenant ID to delete the inactive tenant. A list will show up in the first case for the manager to view the available inactive tenant. After deletion, a message will print out to indicate it is successfully deleted and show an updated tenant list to the manager.

3.3.5 Summarize the Top 10 properties from the favourite properties list.



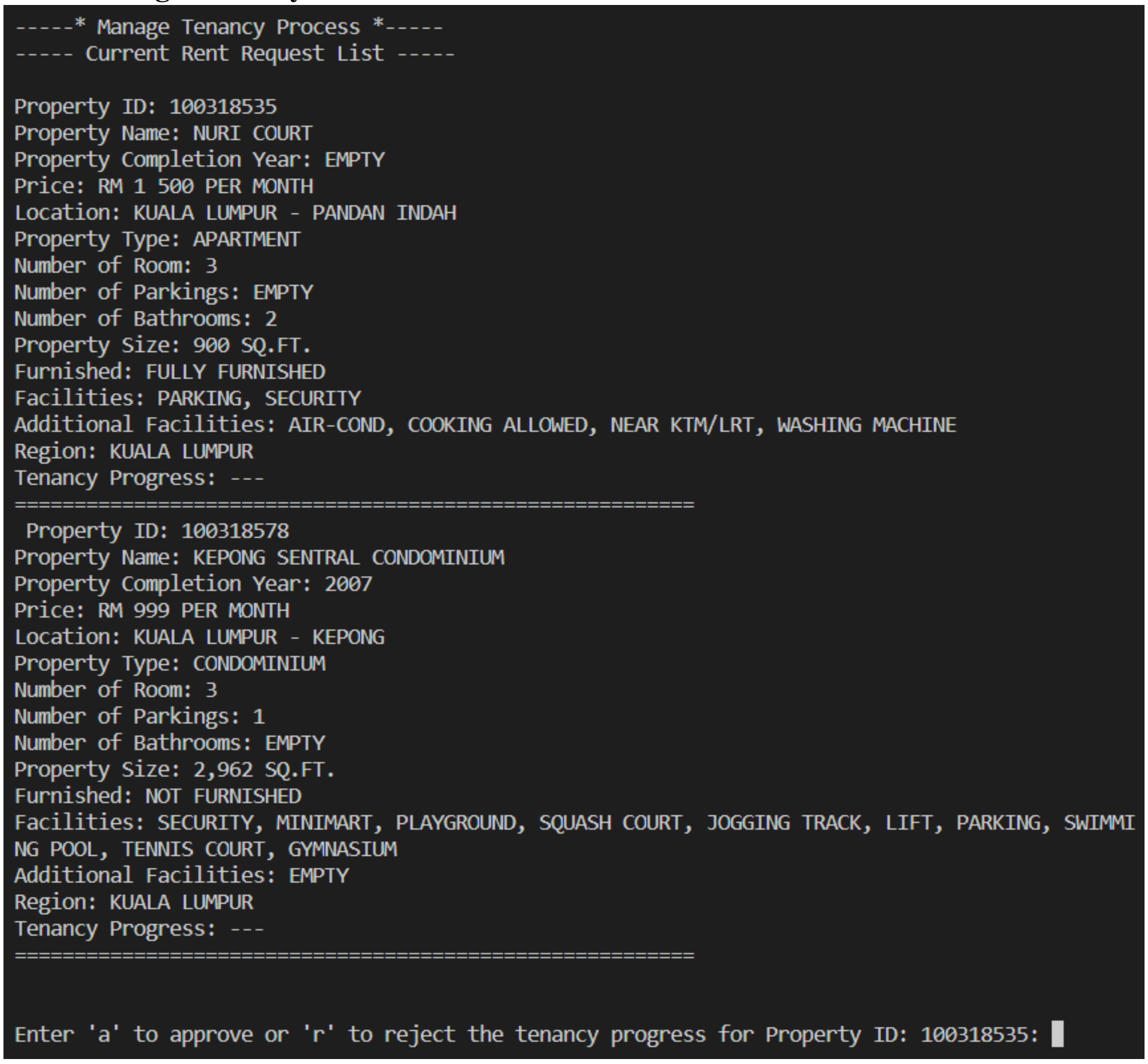
```
Top 6 :The Hipster @ Taman Desa
Property ID: 100318501
Occurrences Count: 3
Location: Kuala Lumpur - Taman Desa
Property Type: Condominium
------------------------
Top 7 :The Park Sky Residence @ Bukit Jalil City
Property ID: 100322813
Occurrences Count: 3
Location: Kuala Lumpur - Bukit Jalil
Property Type: Service Residence
------------------------
Top 8 :Angkasa Impian 1
Property ID: 100198517
Occurrences Count: 2
Location: Kuala Lumpur - Bukit Bintang
Property Type: Condominium
------------------------
Top 9 :Sentrio Suites
Property ID: 100250824
Occurrences Count: 2
Location: Kuala Lumpur - Desa Pandan
Property Type: Service Residence
------------------------
Top 10 :Majestic Maxim
Property ID: 100318198
Occurrences Count: 2
Location: Kuala Lumpur - Cheras
Property Type: Service Residence
------------------------
```

```
Please enter your choice (1-6): 4
----- Top 10 Property Report -----

Number of properties in favouriteListAll: 55

 ----- Top 10 Property Rank from Tenant Favourite List --
Top 1 :Nuri Court
Property ID: 100318535
Occurrences Count: 12
Location: Kuala Lumpur - Pandan Indah
Property Type: Apartment
------------------------
Top 2 :Tiara Mutiara 2
Property ID: 100318196
Occurrences Count: 10
Location: Kuala Lumpur - Old Klang Road
Property Type: Service Residence
------------------------
Top 3 :PV9 Residences @ Taman Melati
Property ID: 100258228
Occurrences Count: 8
Location: Kuala Lumpur - Setapak
Property Type: Service Residence
------------------------
Top 4 :Majestic Maxim
Property ID: 100310024
Occurrences Count: 3
Location: Kuala Lumpur - Cheras
Property Type: Service Residence
------------------------
Top 5 :Mont Kiara Astana
Property ID: 100318448
Occurrences Count: 3
Location: Kuala Lumpur - Mont Kiara
Property Type: Condominium
------------------------
```

Figure 65 Generate the top 10 property functions.

Generate the top 10 properties report from the favourite list for the fourth function. A total of favourite properties will be indicated and followed by the top 10 rank properties and their related detail, as shown above.

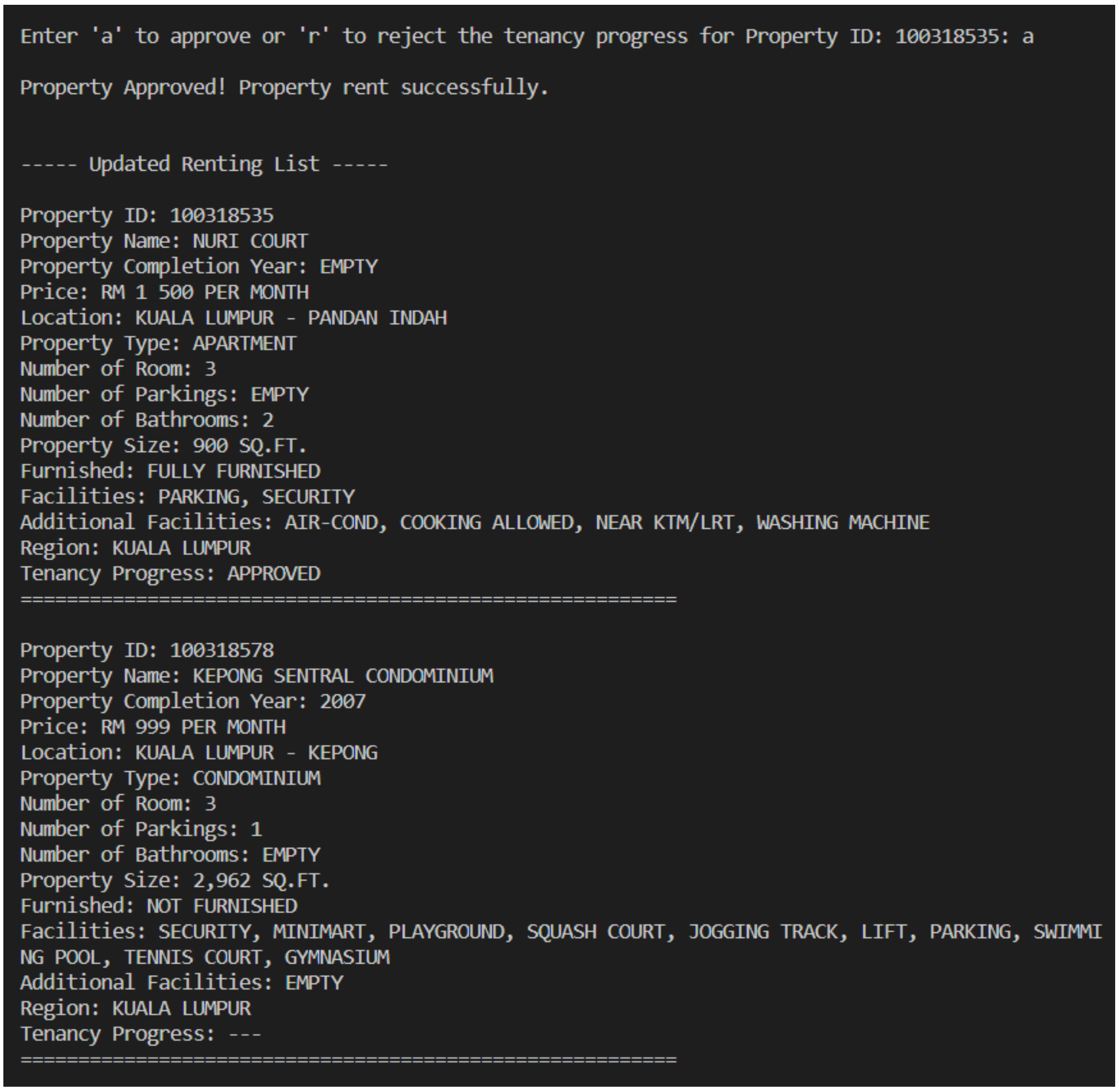
3.3.6 Manage Tenancy Process



```
-----* Manage Tenancy Process *-----
----- Current Rent Request List -----

Property ID: 100318535
Property Name: NURI COURT
Property Completion Year: EMPTY
Price: RM 1 500 PER MONTH
Location: KUALA LUMPUR - PANDAN INDAH
Property Type: APARTMENT
Number of Room: 3
Number of Parkings: EMPTY
Number of Bathrooms: 2
Property Size: 900 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: PARKING, SECURITY
Additional Facilities: AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================
 Property ID: 100318578
Property Name: KEPONG SENTRAL CONDOMINIUM
Property Completion Year: 2007
Price: RM 999 PER MONTH
Location: KUALA LUMPUR - KEPONG
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1
Number of Bathrooms: EMPTY
Property Size: 2,962 SQ.FT.
Furnished: NOT FURNISHED
Facilities: SECURITY, MINIMART, PLAYGROUND, SQUASH COURT, JOGGING TRACK, LIFT, PARKING, SWIMMI
NG POOL, TENNIS COURT, GYMNASIUM
Additional Facilities: EMPTY
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================


Enter 'a' to approve or 'r' to reject the tenancy progress for Property ID: 100318535:
```

Figure 66 Manage tenancy progress function

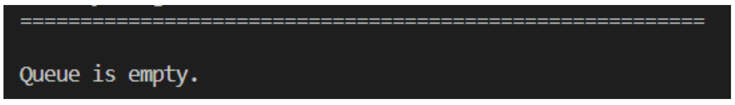
In the fifth function, when the tenant has placed a rent request, the manager must update the tenancy progress. This function shows a queue list of rent requests with property detail and tenancy progress. Input will be required to approve or reject the specific property by inserting ‘a’ or ‘r.’ In default, the tenancy progress is in ‘---.’



```
Enter 'a' to approve or 'r' to reject the tenancy progress for Property ID: 100318535: a

Property Approved! Property rent successfully.


----- Updated Renting List -----

Property ID: 100318535
Property Name: NURI COURT
Property Completion Year: EMPTY
Price: RM 1 500 PER MONTH
Location: KUALA LUMPUR - PANDAN INDAH
Property Type: APARTMENT
Number of Room: 3
Number of Parkings: EMPTY
Number of Bathrooms: 2
Property Size: 900 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: PARKING, SECURITY
Additional Facilities: AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE
Region: KUALA LUMPUR
Tenancy Progress: APPROVED
=========================================================

Property ID: 100318578
Property Name: KEPONG SENTRAL CONDOMINIUM
Property Completion Year: 2007
Price: RM 999 PER MONTH
Location: KUALA LUMPUR - KEPONG
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1
Number of Bathrooms: EMPTY
Property Size: 2,962 SQ.FT.
Furnished: NOT FURNISHED
Facilities: SECURITY, MINIMART, PLAYGROUND, SQUASH COURT, JOGGING TRACK, LIFT, PARKING, SWIMMI
NG POOL, TENNIS COURT, GYMNASIUM
Additional Facilities: EMPTY
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================
```

Figure 67 Manage tenancy progress.

After inserting ‘a’ or ‘r,’ the tenancy progress will be updated to approved or rejected. An updated renting list will be indicated with the updated tenancy progress. The updated list will be store in renting history list.

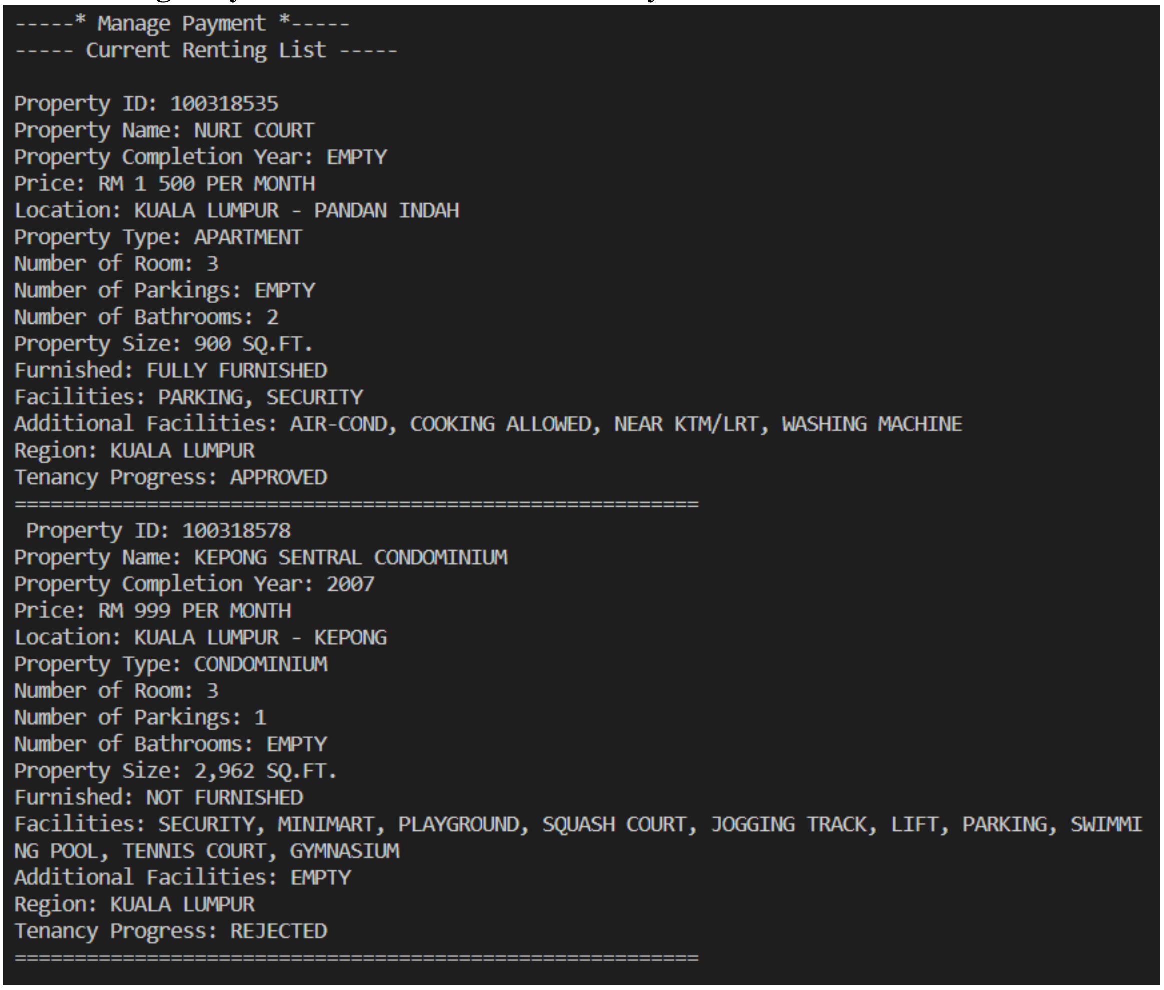


```
=========================================================

Queue is empty.
```

Figure 68 Renting queue list

The system will show ‘Queue is empty’ when all the rent request in the list is completed and managed.

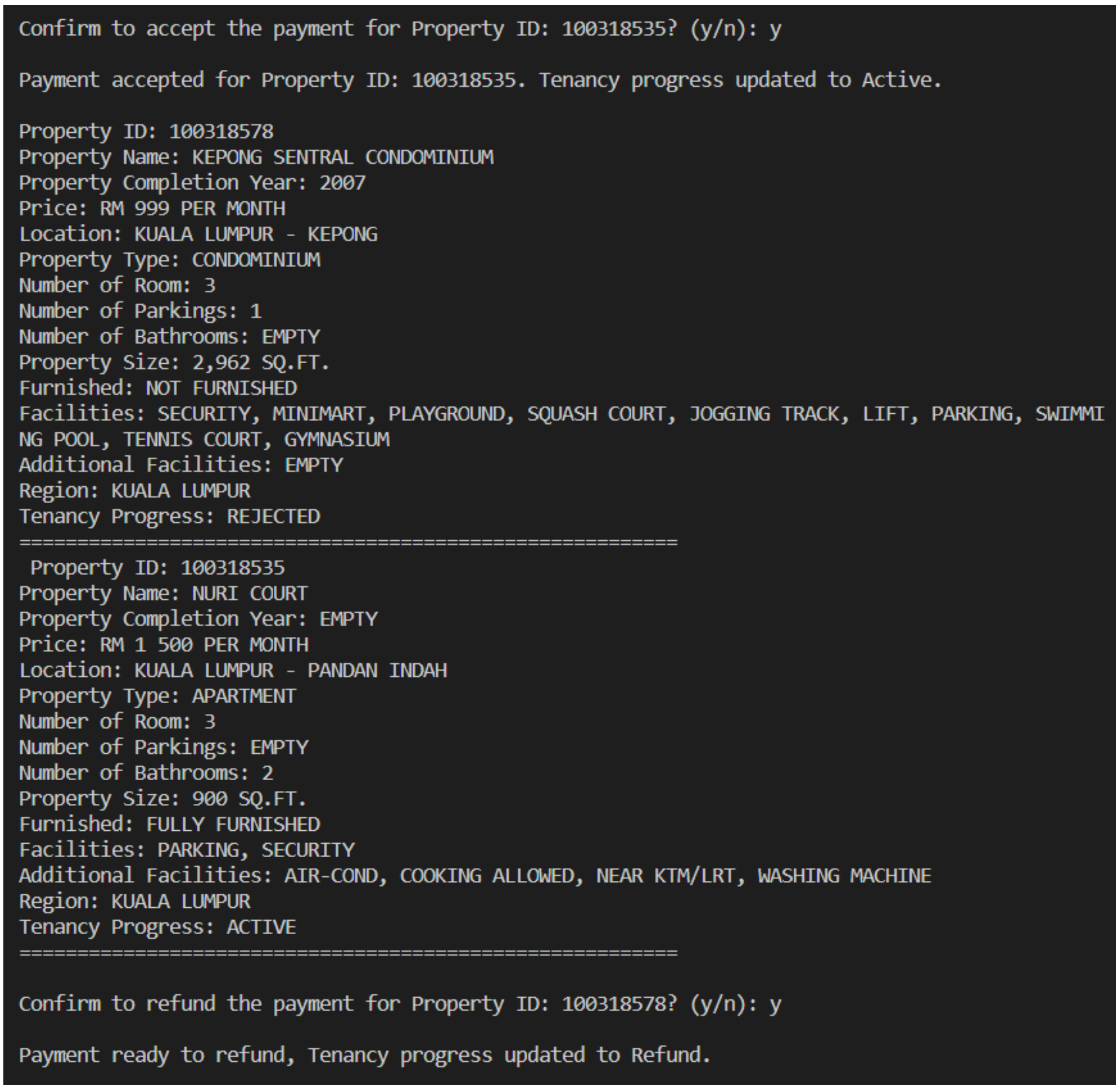
3.3.7 Manage Payment of the confirmed tenancy.



```
-----* Manage Payment *-----
----- Current Renting List -----

Property ID: 100318535
Property Name: NURI COURT
Property Completion Year: EMPTY
Price: RM 1 500 PER MONTH
Location: KUALA LUMPUR - PANDAN INDAH
Property Type: APARTMENT
Number of Room: 3
Number of Parkings: EMPTY
Number of Bathrooms: 2
Property Size: 900 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: PARKING, SECURITY
Additional Facilities: AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE
Region: KUALA LUMPUR
Tenancy Progress: APPROVED
=========================================================
 Property ID: 100318578
Property Name: KEPONG SENTRAL CONDOMINIUM
Property Completion Year: 2007
Price: RM 999 PER MONTH
Location: KUALA LUMPUR - KEPONG
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1
Number of Bathrooms: EMPTY
Property Size: 2,962 SQ.FT.
Furnished: NOT FURNISHED
Facilities: SECURITY, MINIMART, PLAYGROUND, SQUASH COURT, JOGGING TRACK, LIFT, PARKING, SWIMMI
NG POOL, TENNIS COURT, GYMNASIUM
Additional Facilities: EMPTY
Region: KUALA LUMPUR
Tenancy Progress: REJECTED
=========================================================
```

Figure 69 Manage payment function

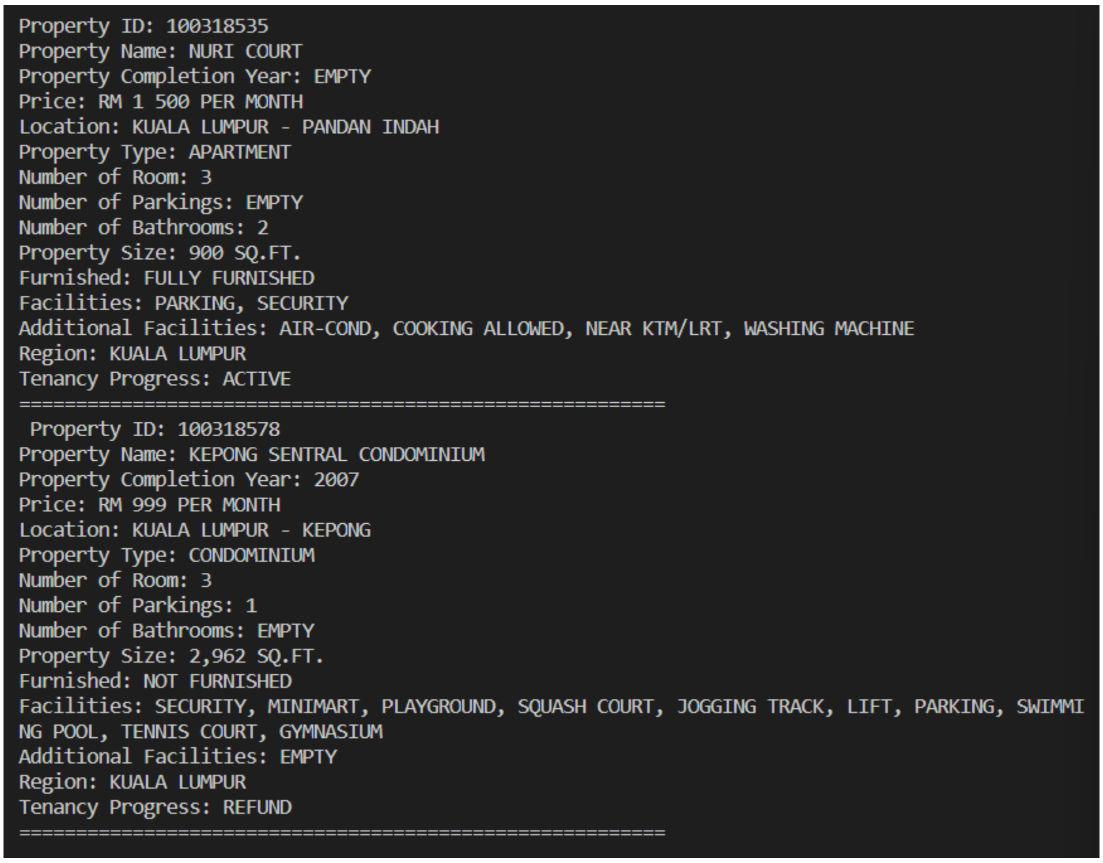
For the sixth function, manage payment once the tenancy had been approved and store in the renting history list. The system will show the updated tenancy progress list with property detail. Managers are able to view whether the specific rent request has been approved or rejected.



```
Confirm to accept the payment for Property ID: 100318535? (y/n): y

Payment accepted for Property ID: 100318535. Tenancy progress updated to Active.

Property ID: 100318578
Property Name: KEPONG SENTRAL CONDOMINIUM
Property Completion Year: 2007
Price: RM 999 PER MONTH
Location: KUALA LUMPUR - KEPONG
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1
Number of Bathrooms: EMPTY
Property Size: 2,962 SQ.FT.
Furnished: NOT FURNISHED
Facilities: SECURITY, MINIMART, PLAYGROUND, SQUASH COURT, JOGGING TRACK, LIFT, PARKING, SWIMMI
NG POOL, TENNIS COURT, GYMNASIUM
Additional Facilities: EMPTY
Region: KUALA LUMPUR
Tenancy Progress: REJECTED
=========================================================
 Property ID: 100318535
Property Name: NURI COURT
Property Completion Year: EMPTY
Price: RM 1 500 PER MONTH
Location: KUALA LUMPUR - PANDAN INDAH
Property Type: APARTMENT
Number of Room: 3
Number of Parkings: EMPTY
Number of Bathrooms: 2
Property Size: 900 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: PARKING, SECURITY
Additional Facilities: AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE
Region: KUALA LUMPUR
Tenancy Progress: ACTIVE
=========================================================

Confirm to refund the payment for Property ID: 100318578? (y/n): y

Payment ready to refund, Tenancy progress updated to Refund.
```

Figure 70 Manager manage payment.

Following, it will require to input whether want to accept the payment for a property or not. If yes, the system will update the related rent request's tenancy progress to active. If the rent request is rejected, the manager will be required to input do want to refund the payment to the tenant. The updated list will stored in renting history list.

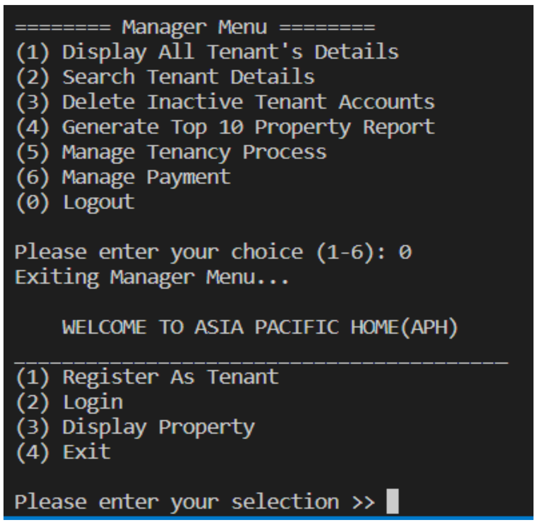


```
Property ID: 100318535
Property Name: NURI COURT
Property Completion Year: EMPTY
Price: RM 1 500 PER MONTH
Location: KUALA LUMPUR - PANDAN INDAH
Property Type: APARTMENT
Number of Room: 3
Number of Parkings: EMPTY
Number of Bathrooms: 2
Property Size: 900 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: PARKING, SECURITY
Additional Facilities: AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE
Region: KUALA LUMPUR
Tenancy Progress: ACTIVE
=========================================================
 Property ID: 100318578
Property Name: KEPONG SENTRAL CONDOMINIUM
Property Completion Year: 2007
Price: RM 999 PER MONTH
Location: KUALA LUMPUR - KEPONG
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 1
Number of Bathrooms: EMPTY
Property Size: 2,962 SQ.FT.
Furnished: NOT FURNISHED
Facilities: SECURITY, MINIMART, PLAYGROUND, SQUASH COURT, JOGGING TRACK, LIFT, PARKING, SWIMMI
NG POOL, TENNIS COURT, GYMNASIUM
Additional Facilities: EMPTY
Region: KUALA LUMPUR
Tenancy Progress: REFUND
=========================================================
```

Figure 71 Updated renting history list

Last an updated list after managing the payment will be visualize to the manager from the renting history list.

3.3.8 Logout



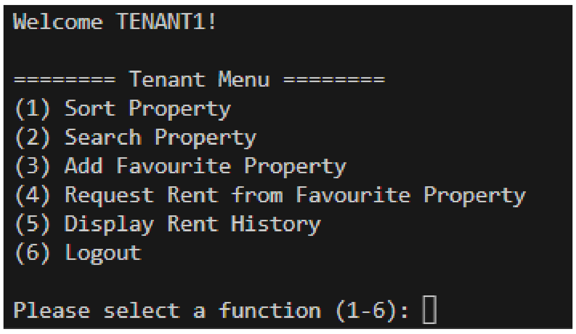
```
======== Manager Menu ========
(1) Display All Tenant's Details
(2) Search Tenant Details
(3) Delete Inactive Tenant Accounts
(4) Generate Top 10 Property Report
(5) Manage Tenancy Process
(6) Manage Payment
(0) Logout

Please enter your choice (1-6): 0
Exiting Manager Menu...

    WELCOME TO ASIA PACIFIC HOME(APH)
__________________________________________
(1) Register As Tenant
(2) Login
(3) Display Property
(4) Exit

Please enter your selection >> 
```

Figure 72: logout function

The logout function will redirect the manager to the system's main menu.

3.4 Tenants

3.4.1 Tenant main menu

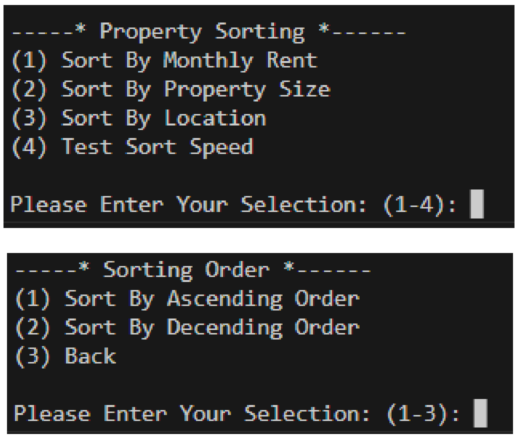


```
Welcome TENANT1!

======== Tenant Menu ========
(1) Sort Property
(2) Search Property
(3) Add Favourite Property
(4) Request Rent from Favourite Property
(5) Display Rent History
(6) Logout

Please select a function (1-6):
```

Figure 73 Tenant Main Menu

As shown in the diagram above, this is the menu for the tenants when they log into the system in the main menu. The tenants are able to select from a few functions in the tenant menu, which are Sort Property, Search Property, Add Favourite Property, Request Rent from Favourite Property and Display Rent History. If they are done with their activities, they can also choose to logout to the main menu.

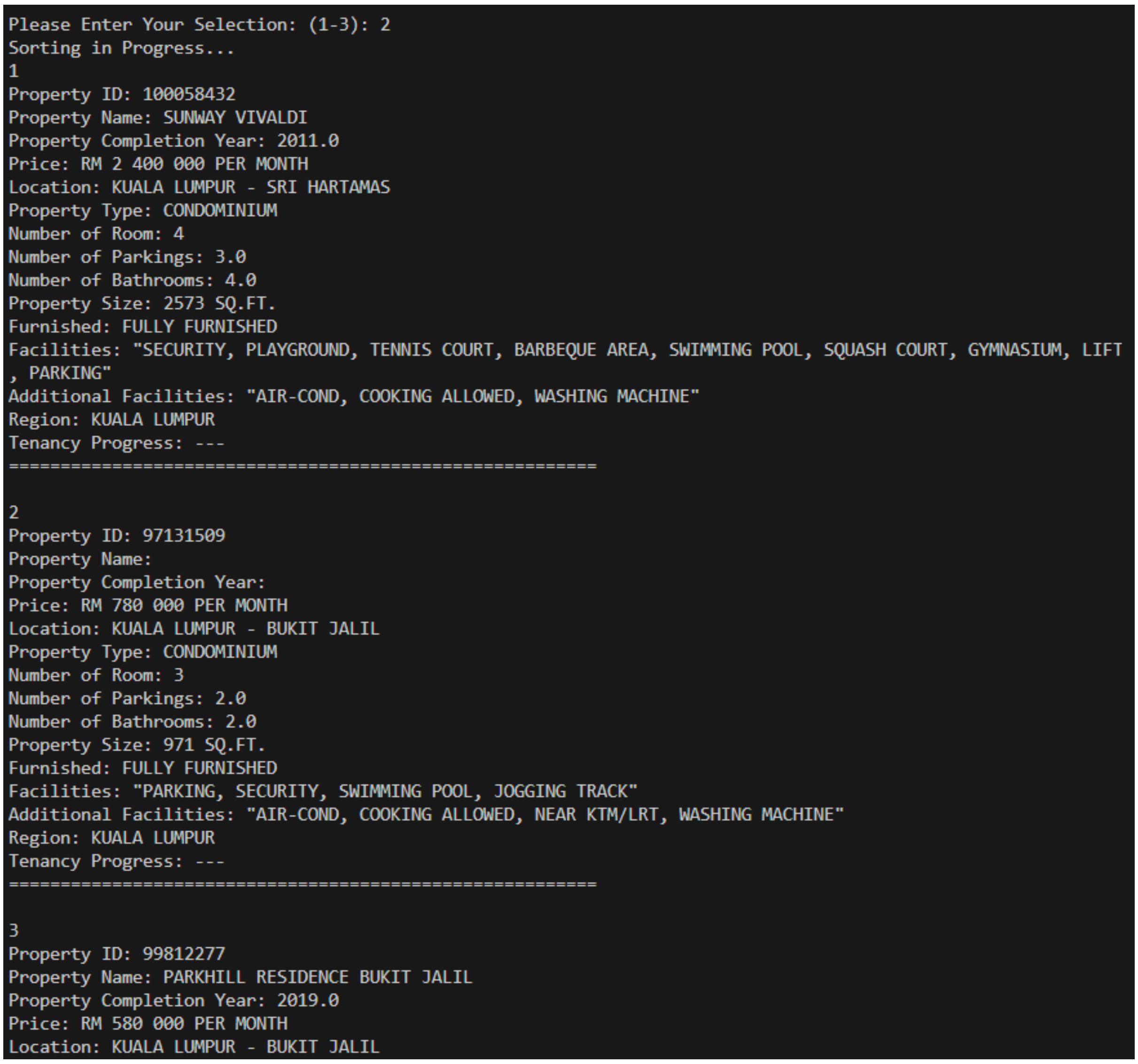
3.4.2 Sort Property



```
-----* Property Sorting *------
(1) Sort By Monthly Rent
(2) Sort By Property Size
(3) Sort By Location
(4) Test Sort Speed

Please Enter Your Selection: (1-4):
```

```
-----* Sorting Order *------
(1) Sort By Ascending Order
(2) Sort By Decending Order
(3) Back

Please Enter Your Selection: (1-3):
```

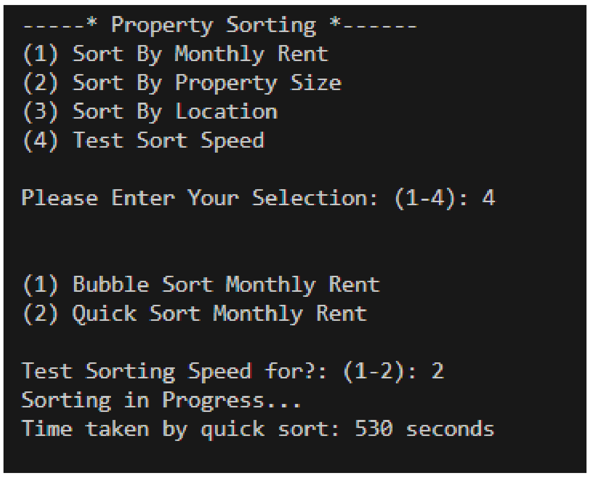
Figure 74 Sort Property Menu



```
Please Enter Your Selection: (1-3): 2
Sorting in Progress...
1
Property ID: 100058432
Property Name: SUNWAY VIVALDI
Property Completion Year: 2011.0
Price: RM 2 400 000 PER MONTH
Location: KUALA LUMPUR - SRI HARTAMAS
Property Type: CONDOMINIUM
Number of Room: 4
Number of Parkings: 3.0
Number of Bathrooms: 4.0
Property Size: 2573 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: "SECURITY, PLAYGROUND, TENNIS COURT, BARBEQUE AREA, SWIMMING POOL, SQUASH COURT, GYMNASIUM, LIFT
, PARKING"
Additional Facilities: "AIR-COND, COOKING ALLOWED, WASHING MACHINE"
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================

2
Property ID: 97131509
Property Name:
Property Completion Year:
Price: RM 780 000 PER MONTH
Location: KUALA LUMPUR - BUKIT JALIL
Property Type: CONDOMINIUM
Number of Room: 3
Number of Parkings: 2.0
Number of Bathrooms: 2.0
Property Size: 971 SQ.FT.
Furnished: FULLY FURNISHED
Facilities: "PARKING, SECURITY, SWIMMING POOL, JOGGING TRACK"
Additional Facilities: "AIR-COND, COOKING ALLOWED, NEAR KTM/LRT, WASHING MACHINE"
Region: KUALA LUMPUR
Tenancy Progress: ---
=========================================================

3
Property ID: 99812277
Property Name: PARKHILL RESIDENCE BUKIT JALIL
Property Completion Year: 2019.0
Price: RM 580 000 PER MONTH
Location: KUALA LUMPUR - BUKIT JALIL
```

Figure 75 Sort Property Results for Monthly Rent Decending

In the Sort Property function, the tenant can select from a few criteria of the properties to sort by, which include the monthly rent, size, and location of the property. The tenants can see to see the properties in ascending order or descending order.

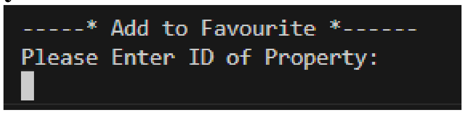


```
-----* Property Sorting *------
(1) Sort By Monthly Rent
(2) Sort By Property Size
(3) Sort By Location
(4) Test Sort Speed

Please Enter Your Selection: (1-4): 4


(1) Bubble Sort Monthly Rent
(2) Quick Sort Monthly Rent

Test Sorting Speed for?: (1-2): 2
Sorting in Progress...
Time taken by quick sort: 530 seconds
```

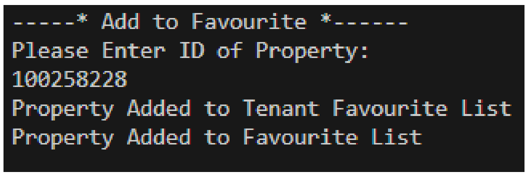
Figure 76 Sorting Time Test

There is also an option for testing the time it takes to sort the properties, but only for the monthly rent criteria in descending order. The time taken to sort will be displayed in seconds.

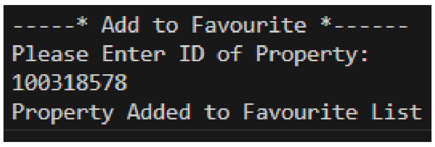
3.4.2 Add favourite property



```
-----* Add to Favourite *------
Please Enter ID of Property:
```



```
-----* Add to Favourite *------
Please Enter ID of Property:
100258228
Property Added to Tenant Favourite List
Property Added to Favourite List
```

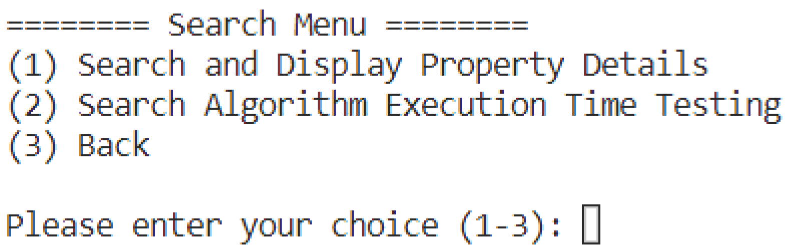


```
-----* Add to Favourite *------
Please Enter ID of Property:
100318578
Property Added to Favourite List
```

Figure 77 Add to Favourite Function

In the Add Favourite Property function, the tenants can use the results from before, either sorting or searching to find their desired property. The tenants can then use the ID of the property to add the property to a favourite list, which opens the possibility for the tenant to request to rent the property. If the Id of the selected property is not found in the favouriteListTenant, it will add into both favouriteListAll and favouriteListTenant, else it will only add into favouriteListAll.

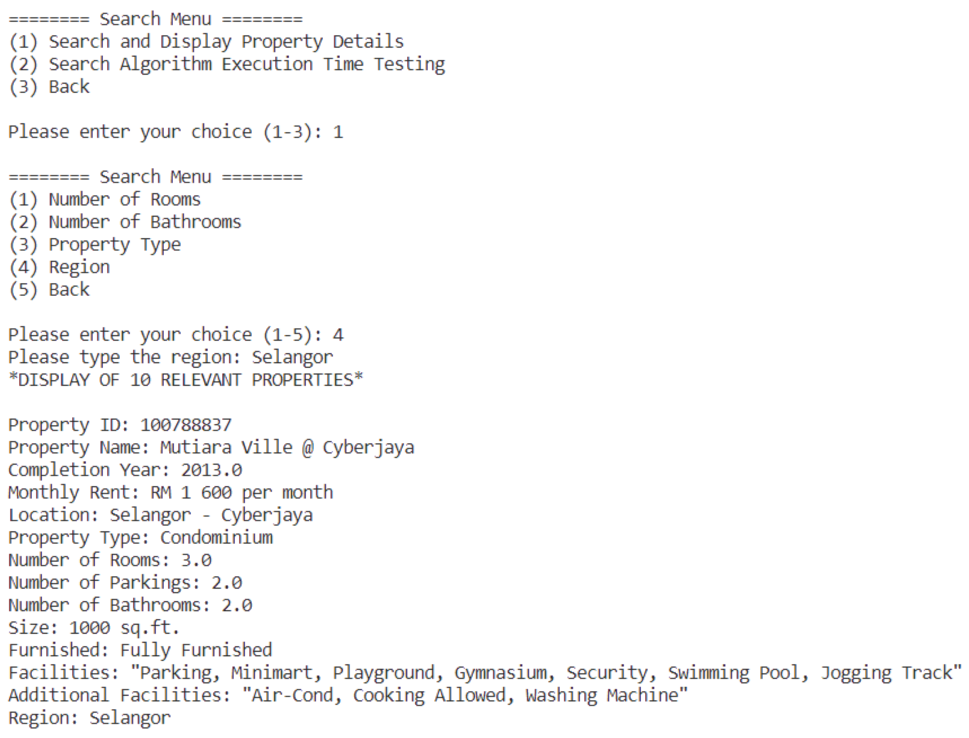
3.4.3 Search Property



```
======== Search Menu ========
(1) Search and Display Property Details
(2) Search Algorithm Execution Time Testing
(3) Back

Please enter your choice (1-3):
```

Figure 78: Search Property Menu

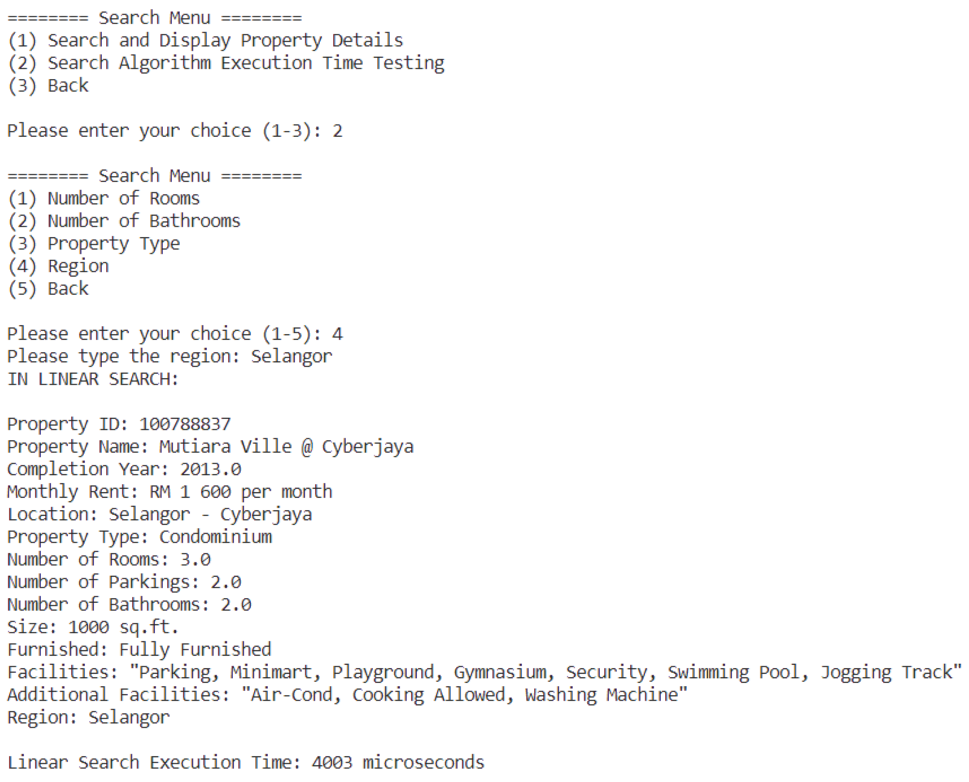
In Search Property Menu, tenant can choose to search and display property details or perform time testing for search algorithm.



```
======== Search Menu ========
(1) Search and Display Property Details
(2) Search Algorithm Execution Time Testing
(3) Back

Please enter your choice (1-3): 1

======== Search Menu ========
(1) Number of Rooms
(2) Number of Bathrooms
(3) Property Type
(4) Region
(5) Back

Please enter your choice (1-5): 4
Please type the region: Selangor
*DISPLAY OF 10 RELEVANT PROPERTIES*

Property ID: 100788837
Property Name: Mutiara Ville @ Cyberjaya
Completion Year: 2013.0
Monthly Rent: RM 1 600 per month
Location: Selangor - Cyberjaya
Property Type: Condominium
Number of Rooms: 3.0
Number of Parkings: 2.0
Number of Bathrooms: 2.0
Size: 1000 sq.ft.
Furnished: Fully Furnished
Facilities: "Parking, Minimart, Playground, Gymnasium, Security, Swimming Pool, Jogging Track"
Additional Facilities: "Air-Cond, Cooking Allowed, Washing Machine"
Region: Selangor
```

Figure 79: Search and Display Property Details function

If choosing for search and display, tenant will be asked to search and display based on number of rooms, number of bathrooms, property type or region. Display of 10 properties with relevant result will be shown based on tenant decision.

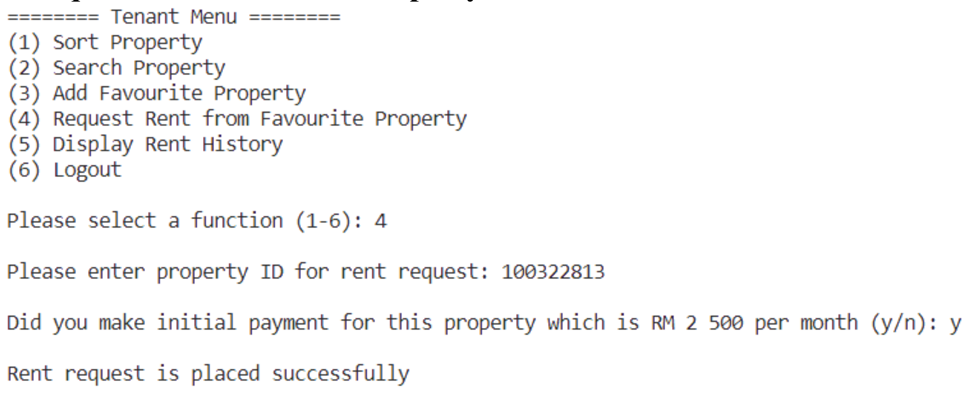


```
======== Search Menu ========
(1) Search and Display Property Details
(2) Search Algorithm Execution Time Testing
(3) Back

Please enter your choice (1-3): 2

======== Search Menu ========
(1) Number of Rooms
(2) Number of Bathrooms
(3) Property Type
(4) Region
(5) Back

Please enter your choice (1-5): 4
Please type the region: Selangor
IN LINEAR SEARCH:

Property ID: 100788837
Property Name: Mutiara Ville @ Cyberjaya
Completion Year: 2013.0
Monthly Rent: RM 1 600 per month
Location: Selangor - Cyberjaya
Property Type: Condominium
Number of Rooms: 3.0
Number of Parkings: 2.0
Number of Bathrooms: 2.0
Size: 1000 sq.ft.
Furnished: Fully Furnished
Facilities: "Parking, Minimart, Playground, Gymnasium, Security, Swimming Pool, Jogging Track"
Additional Facilities: "Air-Cond, Cooking Allowed, Washing Machine"
Region: Selangor

Linear Search Execution Time: 4003 microseconds
```

Figure 80: Search Algorithm Time Testing function

If choosing for time testing, tenant will be asked to test search algorithm based on number of rooms, number of bathrooms, property type or region. Executive of linear search and binary search will be recorded and displayed.

3.4.4 Rent Request from Favourite Property

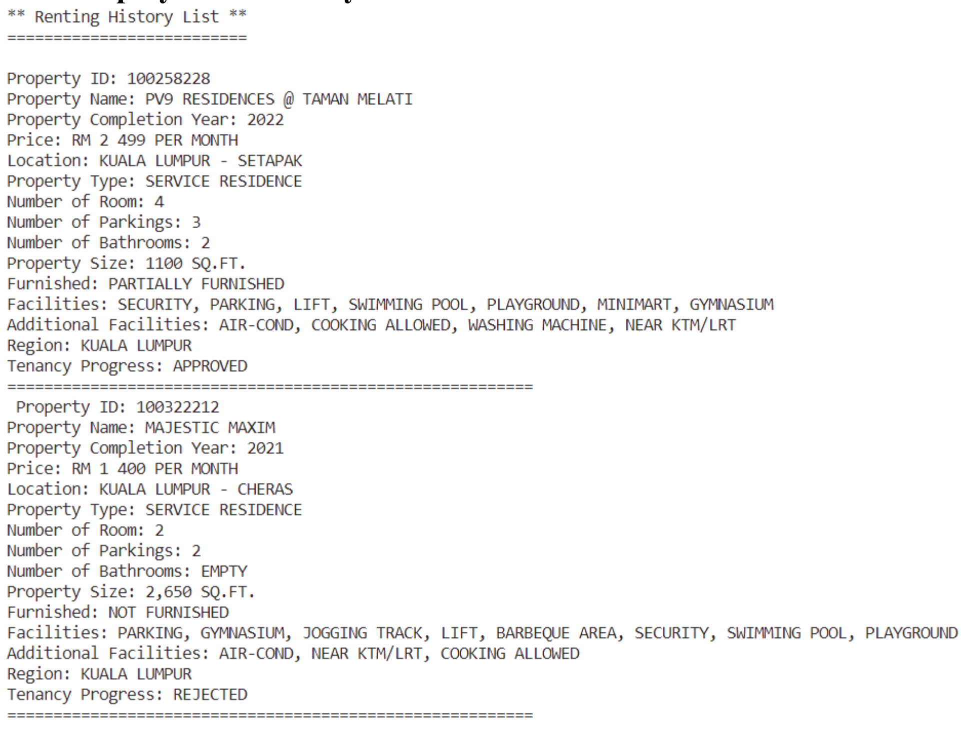


```
======== Tenant Menu ========
(1) Sort Property
(2) Search Property
(3) Add Favourite Property
(4) Request Rent from Favourite Property
(5) Display Rent History
(6) Logout

Please select a function (1-6): 4

Please enter property ID for rent request: 100322813

Did you make initial payment for this property which is RM 2 500 per month (y/n): y

Rent request is placed successfully
```

Figure 81: Function for Rent Request from Favourite Property

For rent request, tenant needs to insert property ID in which that the property must be from favourite property list. After verifying the property id is from favourite list, tenant will be asked whether he has made initial payment for the specific property. The specific property will be placed into rent request list after confirmation of initial payment.

3.4.5 Display Rent History List



```
** Renting History List **
==========================

Property ID: 100258228
Property Name: PV9 RESIDENCES @ TAMAN MELATI
Property Completion Year: 2022
Price: RM 2 499 PER MONTH
Location: KUALA LUMPUR - SETAPAK
Property Type: SERVICE RESIDENCE
Number of Room: 4
Number of Parkings: 3
Number of Bathrooms: 2
Property Size: 1100 SQ.FT.
Furnished: PARTIALLY FURNISHED
Facilities: SECURITY, PARKING, LIFT, SWIMMING POOL, PLAYGROUND, MINIMART, GYMNASIUM
Additional Facilities: AIR-COND, COOKING ALLOWED, WASHING MACHINE, NEAR KTM/LRT
Region: KUALA LUMPUR
Tenancy Progress: APPROVED
========================================================
 Property ID: 100322212
Property Name: MAJESTIC MAXIM
Property Completion Year: 2021
Price: RM 1 400 PER MONTH
Location: KUALA LUMPUR - CHERAS
Property Type: SERVICE RESIDENCE
Number of Room: 2
Number of Parkings: 2
Number of Bathrooms: EMPTY
Property Size: 2,650 SQ.FT.
Furnished: NOT FURNISHED
Facilities: PARKING, GYMNASIUM, JOGGING TRACK, LIFT, BARBEQUE AREA, SECURITY, SWIMMING POOL, PLAYGROUND
Additional Facilities: AIR-COND, NEAR KTM/LRT, COOKING ALLOWED
Region: KUALA LUMPUR
Tenancy Progress: REJECTED
========================================================
```

Figure 82: Rent Request List Display function

Tenant can display his rent history list of properties after tenancy process and payment process of the requested properties have been managed by manager.

4.0 Conclusion

In conclusion, the Asia Pacific Home System was able to be delivered as a software system that can make the process for Klang Valley customers that wish to rent homes easier by executing all the available functions in our system for three roles of user. The software system can accommodate the potential users of the system, which are the tenants, managers, and admins by delivering the appropriate functions that are required and additional features as important as, the data structure, class, and algorithm implementation in the system for more advance utilizing. In short, this assignment has been an eye-opening experience in terms of learning the practical implementations of the various types of data structures. We were able to implement doubly linked lists as the core data structure to store the data required by the system and use queue data structure for a different use case. We also involved multiple classes to store the specific user header. As for the sorting algorithm, we involved quick sort and binary sort, while the searching algorithm is linear search and binary search. Throughout the system, we had consolidated the data structure knowledge under the guidance of our lecturer and also accomplished the objective and requirement for the system from our effort.

4.1 Summary with Limitation

There are various shortcomings in the system that need to be considered for upcoming upgrades. First off, users looking for specific property information may be limited by the property search functionality's lack of a direct search option using property IDs. This lack of quick access may cause searches to take longer time and lower customer satisfaction. Second, while the queue data structure ensures fairness in function manage payments and tenancy progress, it may prevent property managers from effectively responding to urgent or out-of-order updates. This might make it harder for them to give rent demands top priority. Finally, the need for renters to make payments after submitting a rent request presents an unanticipated restriction: unavoidable events may make a rented property unusable after payment. Tenant dissatisfaction may result from this circumstance and may require additional work to rectify, which may influence tenants' levels of satisfaction and confidence. By addressing these restrictions, the user experience can be enhanced, and the system's adaptability can be increased.

4.2 Future Work

Future improvements to the property management system have tremendous potential to improve both its usability and functionality. To manage payment procedures more efficiently, one stimulating path is using a doubly linked list structure. Managers can apply search to manage the payment. The system may effortlessly integrate payments with relevant tenant information by integrating this structure, giving property managers a thorough view of financial transactions, and enabling more precise payment management. Streamlining the rent request procedure could improve the system's communication with tenants. This calls for streamlining the process from the tenant's initial inquiry about a rental until the last payment is approved. The management can approve the request after confirming that the property is available, which will then require the tenant to start the payment procedure. Other than that, the search and show property detail function could allow the user to search by ID for more convenience.

These developments have the potential to greatly streamline operations, enhance tenant-manager communication, and boost overall efficiency in the dynamic world of property management systems. The system can create the groundwork for a more simplified, automated, and user-friendly approach to property management that benefits both renters and property managers by utilizing the capabilities of a doubly linked list structure, enhancing the search property features, and improving the rent request procedure.

4.3 Reflection

A number of insightful and useful lessons have been learned from the creation and use of the property management system. First and foremost, a deeper comprehension of data structures and their useful applications has been gained because the system effectively manages property and payment information by using ideas like doubly linked lists and queues. Furthermore, the importance of user-centered design has been brought to light through the development process, even if it is working on the console tab. It is crucial to provide tidiness that is simple to use and intuitive so users can work with the system effectively. Besides that, the system's valuable function also helps us to consolidate our C++ programming skills and knowledge. Learn programming manners and habits. In short, creating the APH property management system has improved technical proficiency while providing a comprehensive understanding of system architecture, user experience, and error handling in a real-world setting. These teachings apply to various other projects, problem-solving situations, and software development.

5.0 References

Abba, I. V. (17 March, 2023). *Binary Search in C++ – Algorithm Example*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/binary-search-in-c-algorithm-example/

CodesCracker. (2023). *C++ Program for Linear Search*. Retrieved from CodesCracker: https://codescracker.com/cpp/program/cpp-program-linear-search.htm

Programiz. (2023). *Binary Search*. Retrieved from Programiz: https://www.programiz.com/dsa/binary-search

QuickSort – Data Structure and Algorithm Tutorials. (9 August, 2023). Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/quick-sort/

Sharma, S. (22 November, 2019). *Binary Insertion Sort in C++*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/binary-insertion-sort-in-cplusplus#:~:text=Insertion%20sort%20is%20sorting%20technique,array%20for%20finding%20the%20element.