Faculty of Computer & Information Sciences
Ain Shams University
Subject: CSW150
Fundamentals of Structured Programming

Examiners: Prof. Zaki Taha
Dr. Yasmine Afify
Dr. Salsabil Amin
Academic year: 2nd term 2019-2020
Year: 1st undergraduate

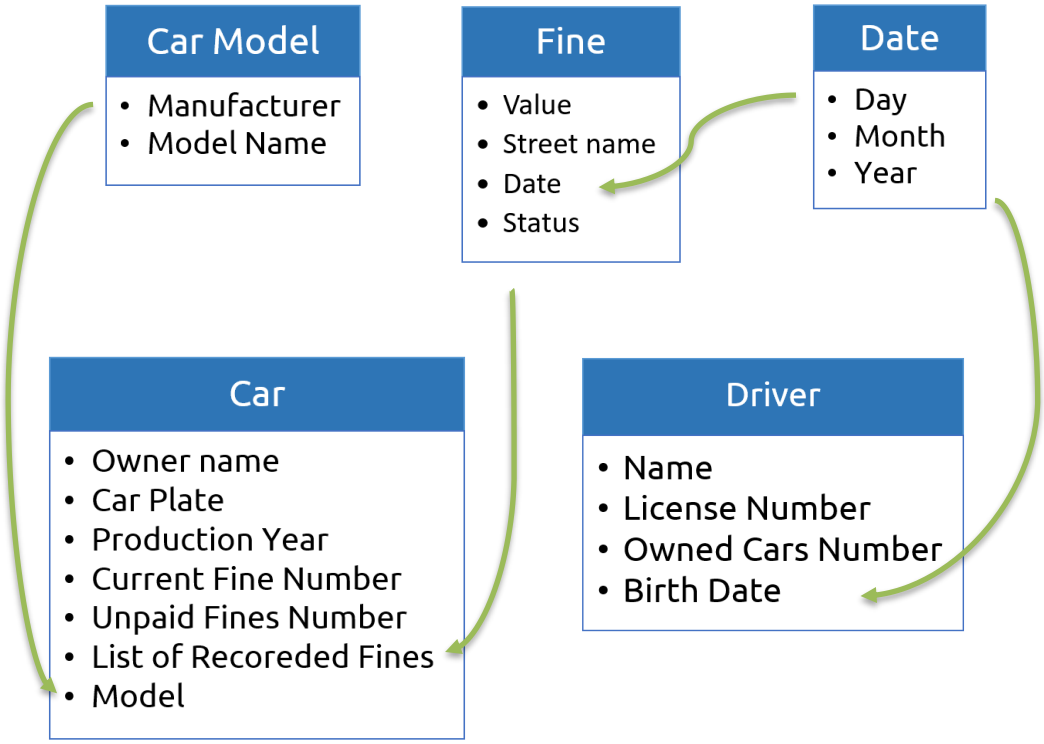
Research Topic Version (D)

Title: Traffic Control System

تحذير هام: علي الطالب عدم كتابة اسمه أو كتابة اي شيء يدل على شخصيته

1. Data Model

System Structs:



(Entity Relationship Diagram)

1. `struct date`

Constructing a date datatype to hold the day, month and year

- Using `unsigned short int` with day, month and year, as none of them would **exceed** the `short int` range, and `unsigned`, as not all of them cannot be negative.

---

2. `struct model`

Constructing a model datatype to hold the car model that consists of the manufacturer name and the model name

- Using `string` to store the manufacturer name and the model name as it can hold whatever characters they are consisted of, whether they are numbers or from the Latin alphabetical.

---

3. `struct fine`

Constructing a fine datatype to hold the fine information, value, the street in which it was committed, status.

- Using `float` for the value as it the speed could be a floating-point value so the fine value can be as well.
- Using `string` for the street name as it can hold a whole sentence representing the street name.
- Using the `date` datatype to store the date of the fine.
- Using a `bool` type to store it status, whether the fine is paid or not.

4. `struct car`

Constructing a car datatype to hold the car information, the owner of the car name, the car plate, the production year, the total amount of fines in the car history, and a list to hold and keep track of these fines, the car model and the current amount of unpaid fines.

- Using `string` for the owner name as it can hold their whole name.
- Using `unsigned int` for the car plate, production year, current fine number and the current amount of unpaid fines, `unsigned` as they all cannot be negative and they fit right in the `int` datatype range.
- Using `array` of `fine` datatype to store the car fines giving it size `MAX_FINES_NUMBER`, which can be easily set and modified in the `#define` section for the constants.
- Using `model` datatype to store the car model.

---

5. `struct driver`

Constructing a driver datatype to hold the driver information, their name, their license number, their birth date, the number of owned cars.

- Using `unsigned int` for the license number and the number of owned cars, `unsigned` as neither of both can be negative and they fit right in the `int` datatype range.
- Using `string` for the owner name as it can hold their whole name.
- Using the `date` datatype to store the driver's birth date.

System Constants:

All system constants are declared in the **global** scope so that all system functions can have **access** to them,

- `MAX_OWNED_CARS`

  Represents the max number of cars that one driver could own.

- `MAX_FINES_NUMBER`

  Represents the max number of fines that one car could have.

- `MAX_DRIVER_CAPACITY`

  Represents the max number of drivers stored in the system.

- `MAX_CARS_NUMBER`

  Represents the max number of cars stored in the system, calculated by multiplying `MAX_OWNED_CARS` by `MAX_DRIVER_CAPACITY.`

- `MIN_MAX_SPEED`

  Represents the minimum speed limit possible in the system.

System Variables:

Apart from the constants mentioned above, all system data is stored in variables as they all can vary and hold different date while processing the user input.

- `unsigned int current_license`

  Tells how many driver registered in the system, used to index the new registered driver, declared globally to be able to keep track of its value and use it in various different functions.

System Arrays:

Declared globally so that the all system functions will have access to their data.

- `driver stored_drivers[MAX_DRIVER_CAPACITY]`

  Using array of `driver` data type to store the drivers registered in the system, setting its size to `MAX_DRIVER_CAPACITY`, which is constant mentioned above that can be easily set and modified.

- `car stored_cars[MAX_CARS_NUMBER]`

  Using array of `car` data type to store the system cars, setting its size to `MAX_CARS_NUMBER`, which is constant mentioned above that, can be easily set and modified.

Instead of making an array of `car` for every driver, storing all cars in one array is much easier for search purposes, with reserving `MAX_OWNED_CARS` number of indices for every driver.

For instance, with `MAX_OWNED_CARS` expands to 3, a driver with index 4 in `stored_drivers` will have the indices 12, 13, 14 reserved for his cars in `stored_cars`.

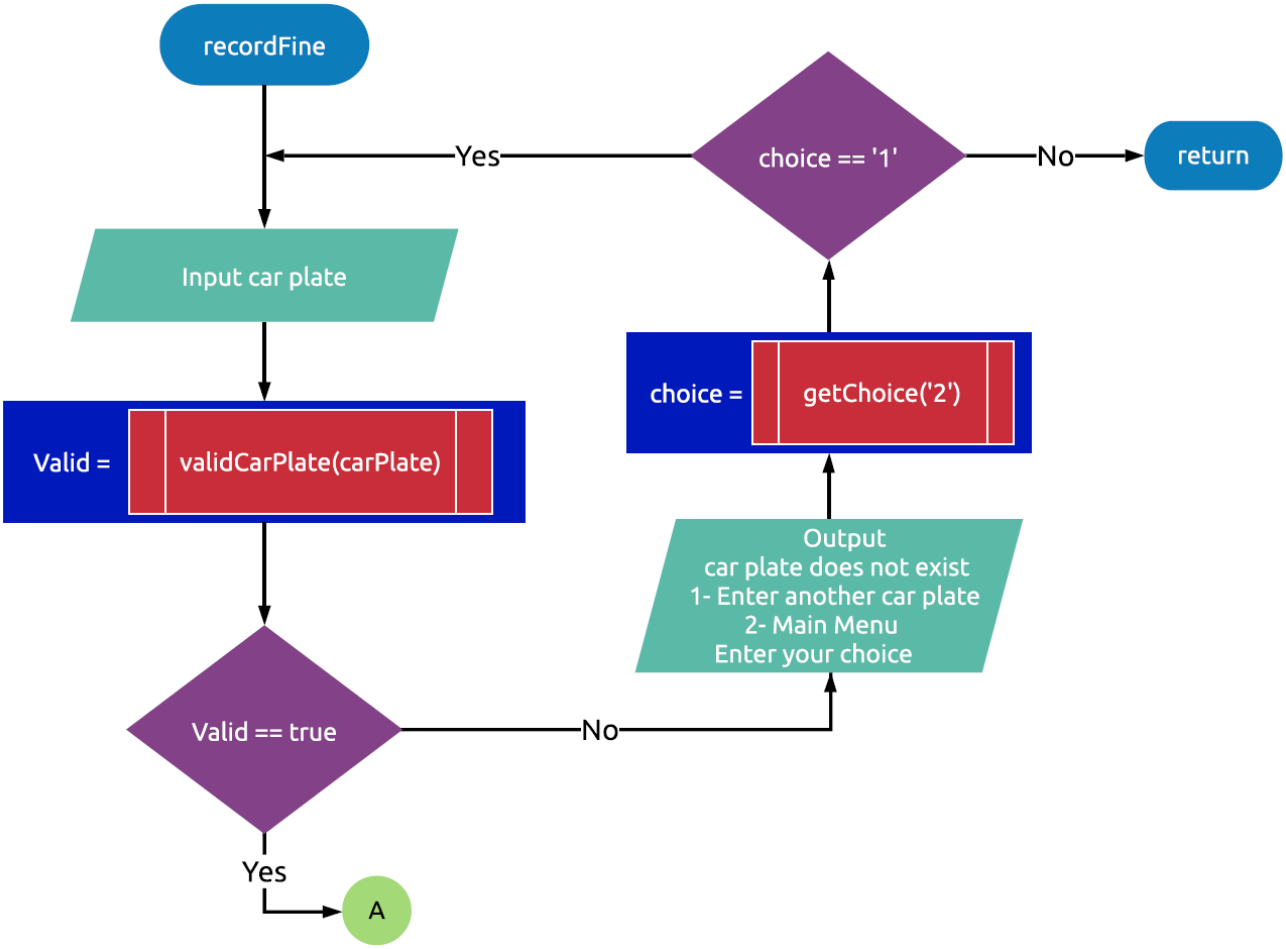
Input Validation:

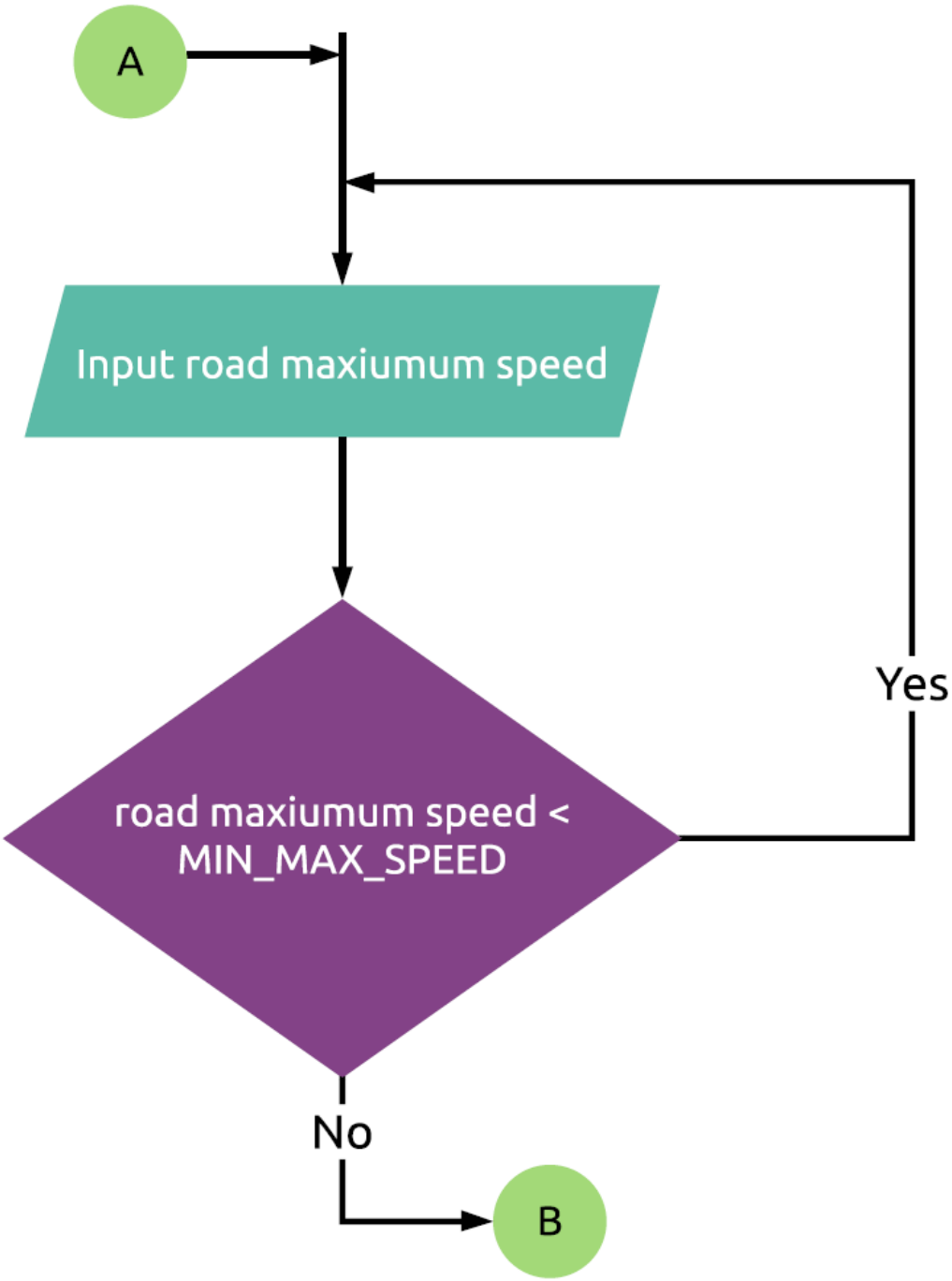
The basic idea to validate the user input is to take their input then put it in an in an infinite loop that only breaks if the input is valid and if it is not, it asks for a new valid input.

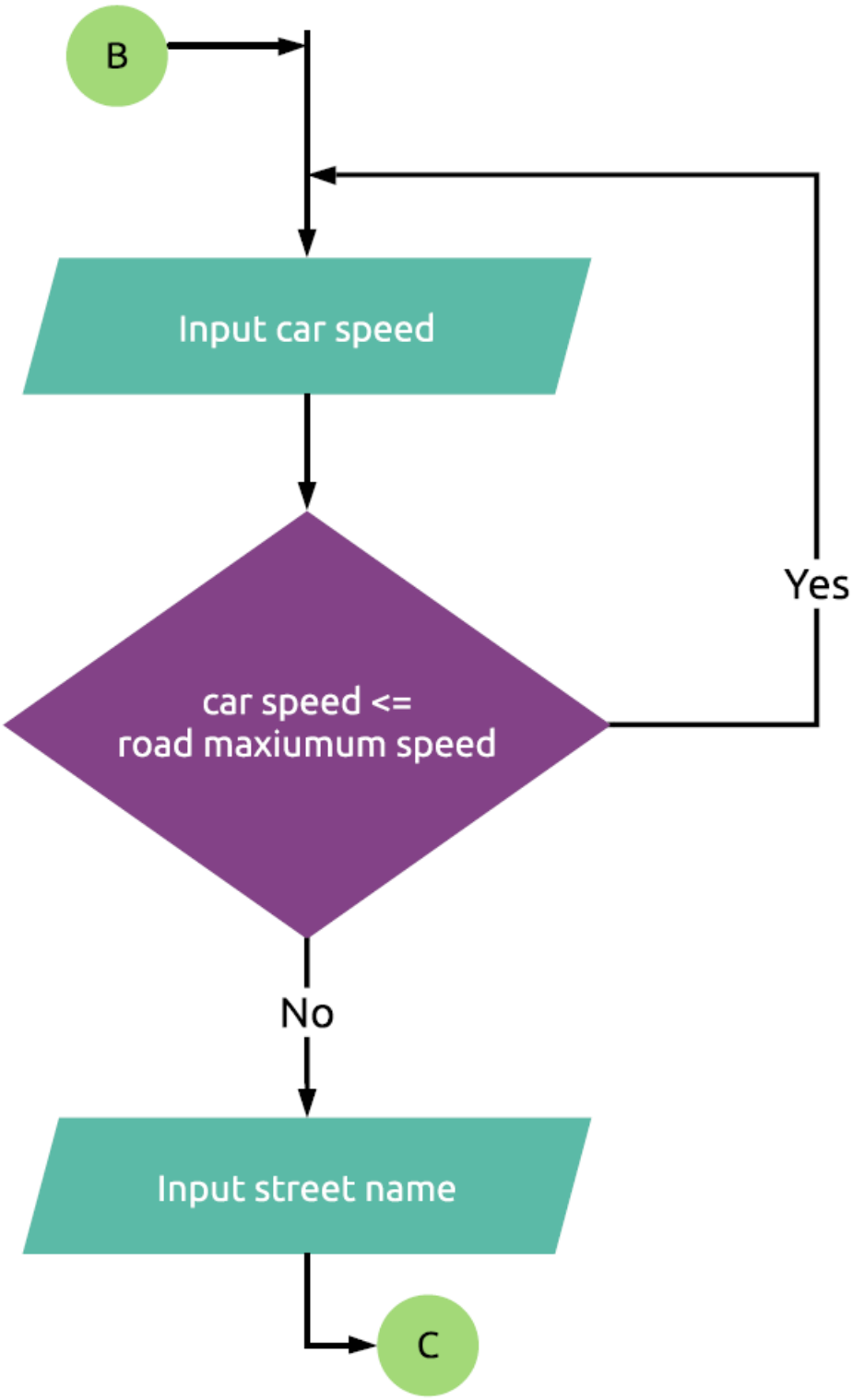
Several functions are used in the system as the break condition to the infinite loop except for few inputs are validated by constants or another inputs.

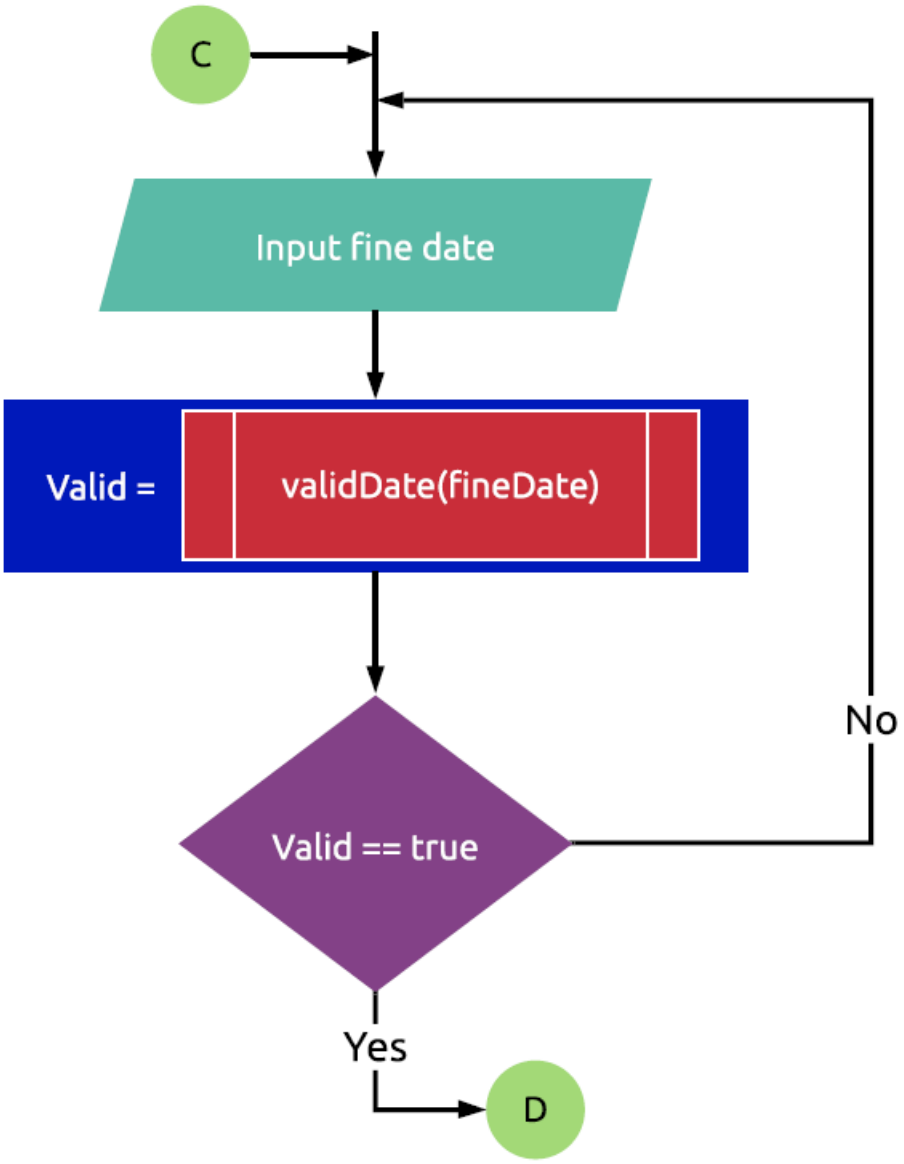
2. Logical Model (Algorithm)

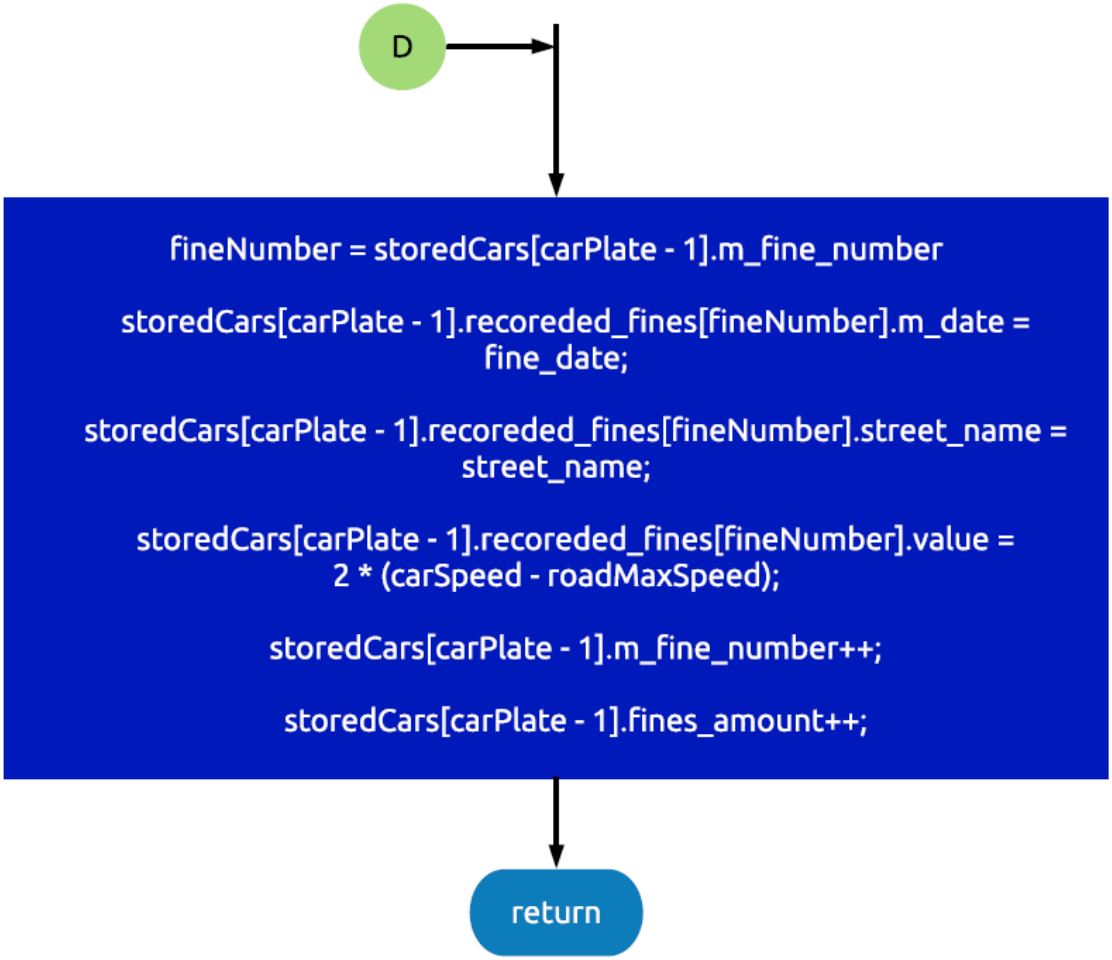
- `void record_fine()`




A
Input road maxiumum speed
road maxiumum speed <
MIN_MAX_SPEED
Yes
No
B

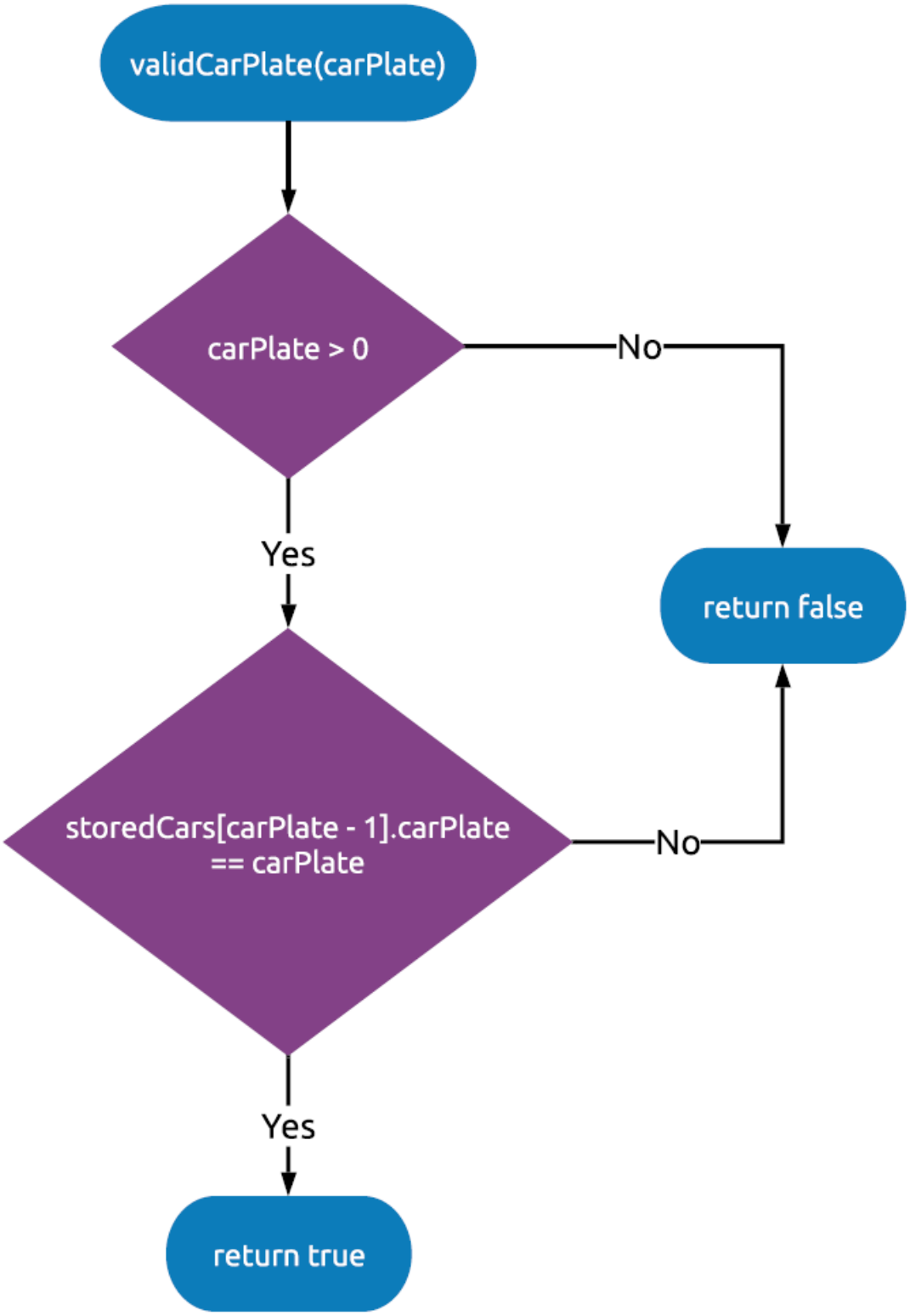

B
Input car speed
car speed <=
road maxiumum speed
Yes
No
Input street name
C


C
Input fine date
Valid =
validDate(fineDate)
Valid == true
No
Yes
D

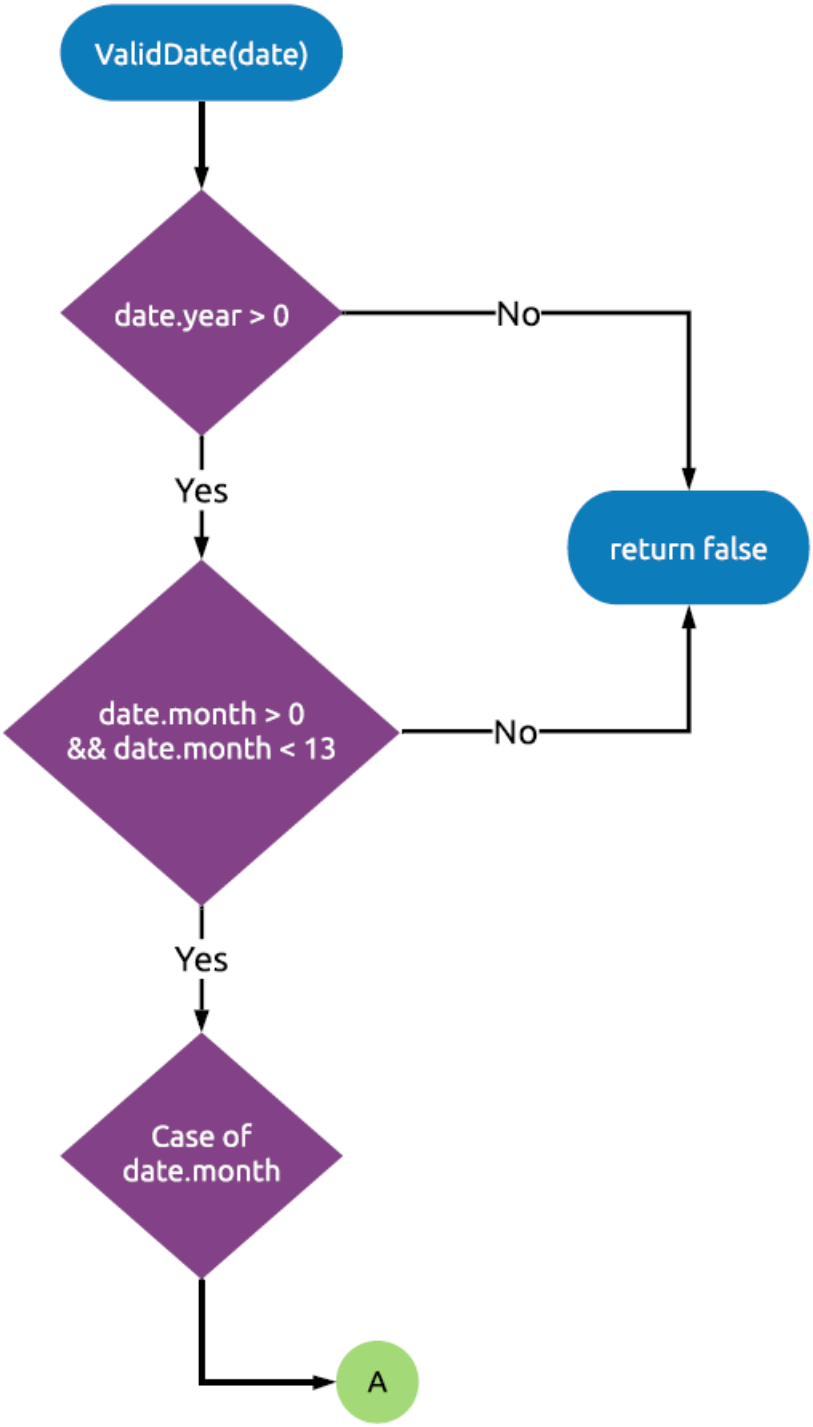

D
fineNumber = storedCars[carPlate - 1].m_fine_number
storedCars[carPlate - 1].recoreded_fines[fineNumber].m_date = fine_date;
storedCars[carPlate - 1].recoreded_fines[fineNumber].street_name = street_name;
storedCars[carPlate - 1].recoreded_fines[fineNumber].value = 2 * (carSpeed - roadMaxSpeed);
storedCars[carPlate - 1].m_fine_number++;
storedCars[carPlate - 1].fines_amount++;
return

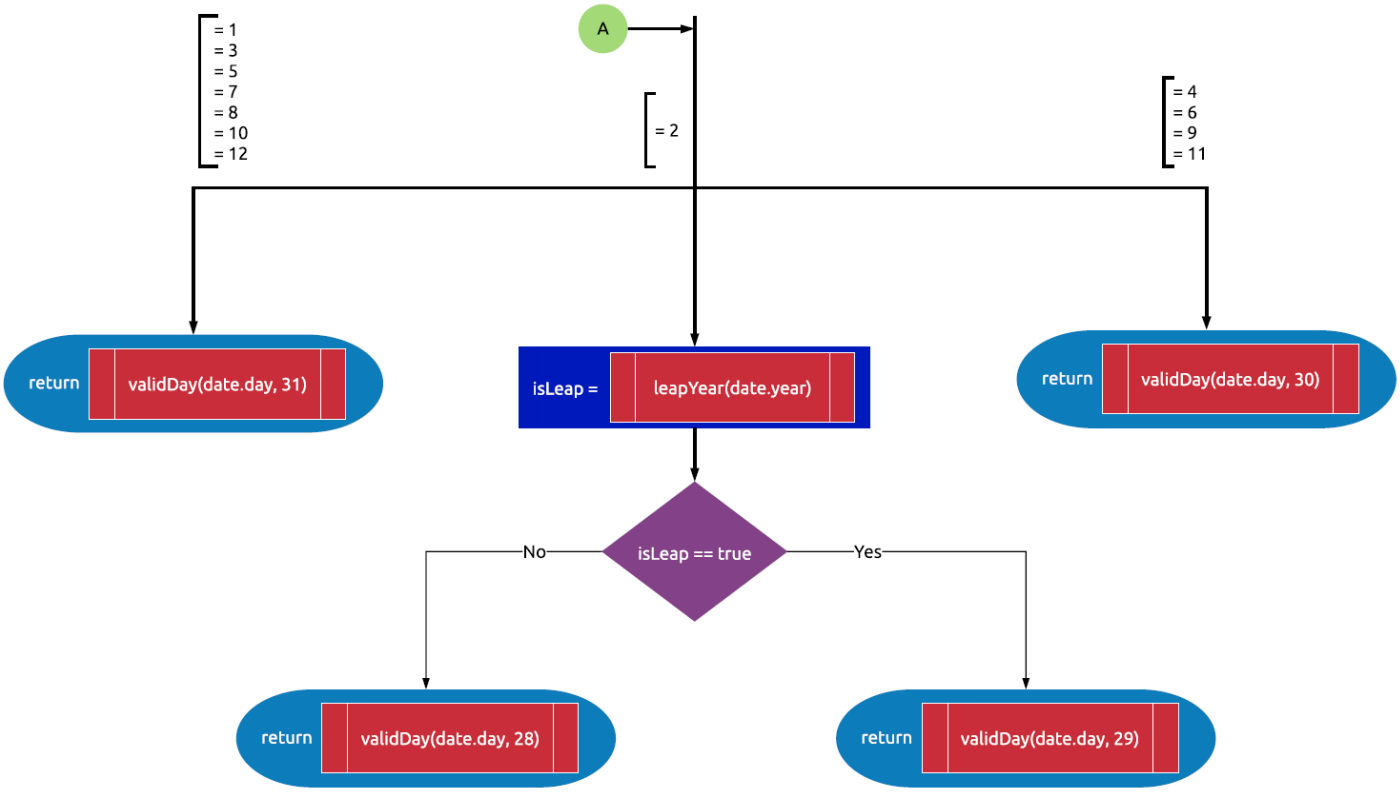
The predefined processes:

- `bool valid_car_plate(unsigned int car_plate)`

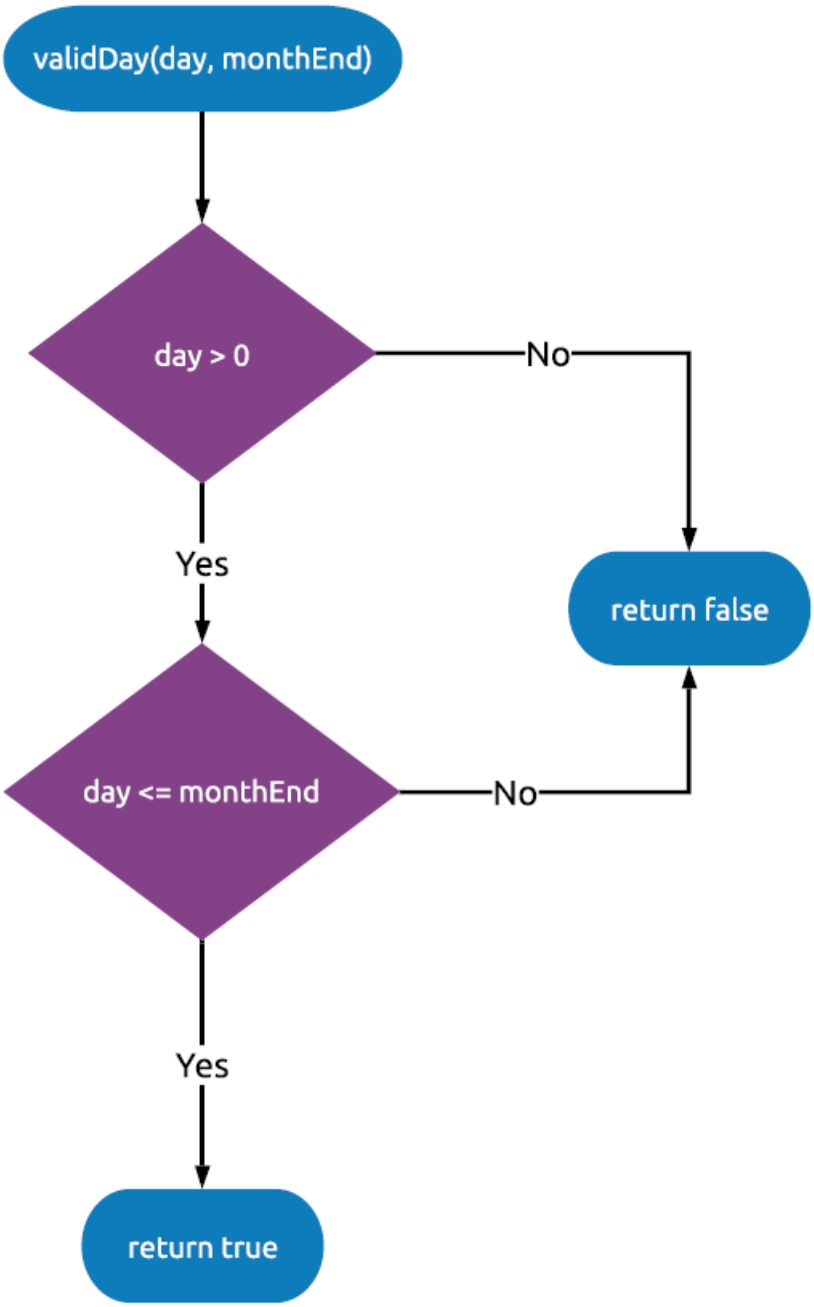


- `bool valid_date(date input_date)`

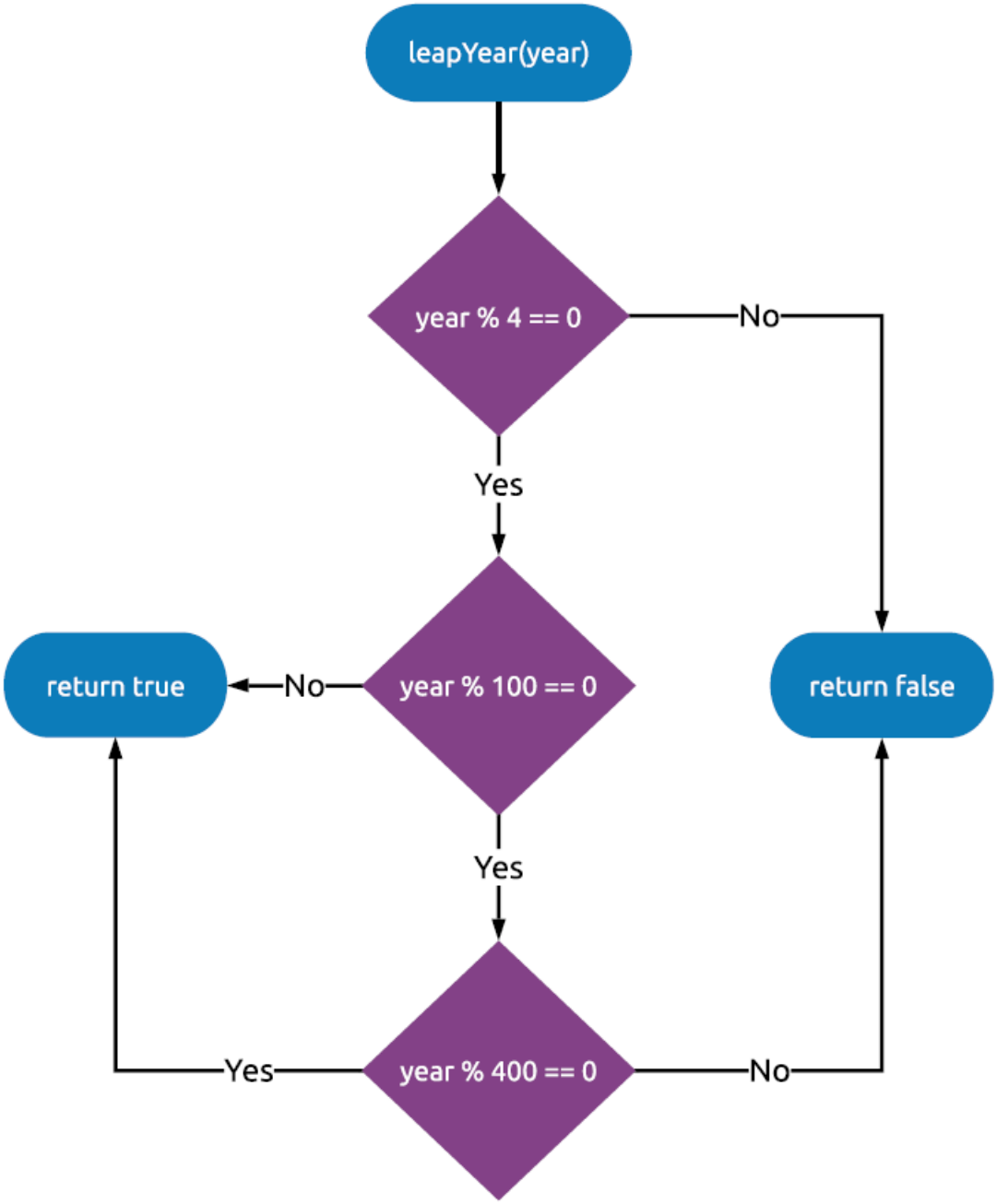



A
= 1
= 3
= 5
= 7
= 8
= 10
= 12
= 2
= 4
= 6
= 9
= 11
return
validDay(date.day, 31)
isLeap =
leapYear(date.year)
return
validDay(date.day, 30)
isLeap == true
No
Yes
return
validDay(date.day, 28)
return
validDay(date.day, 29)

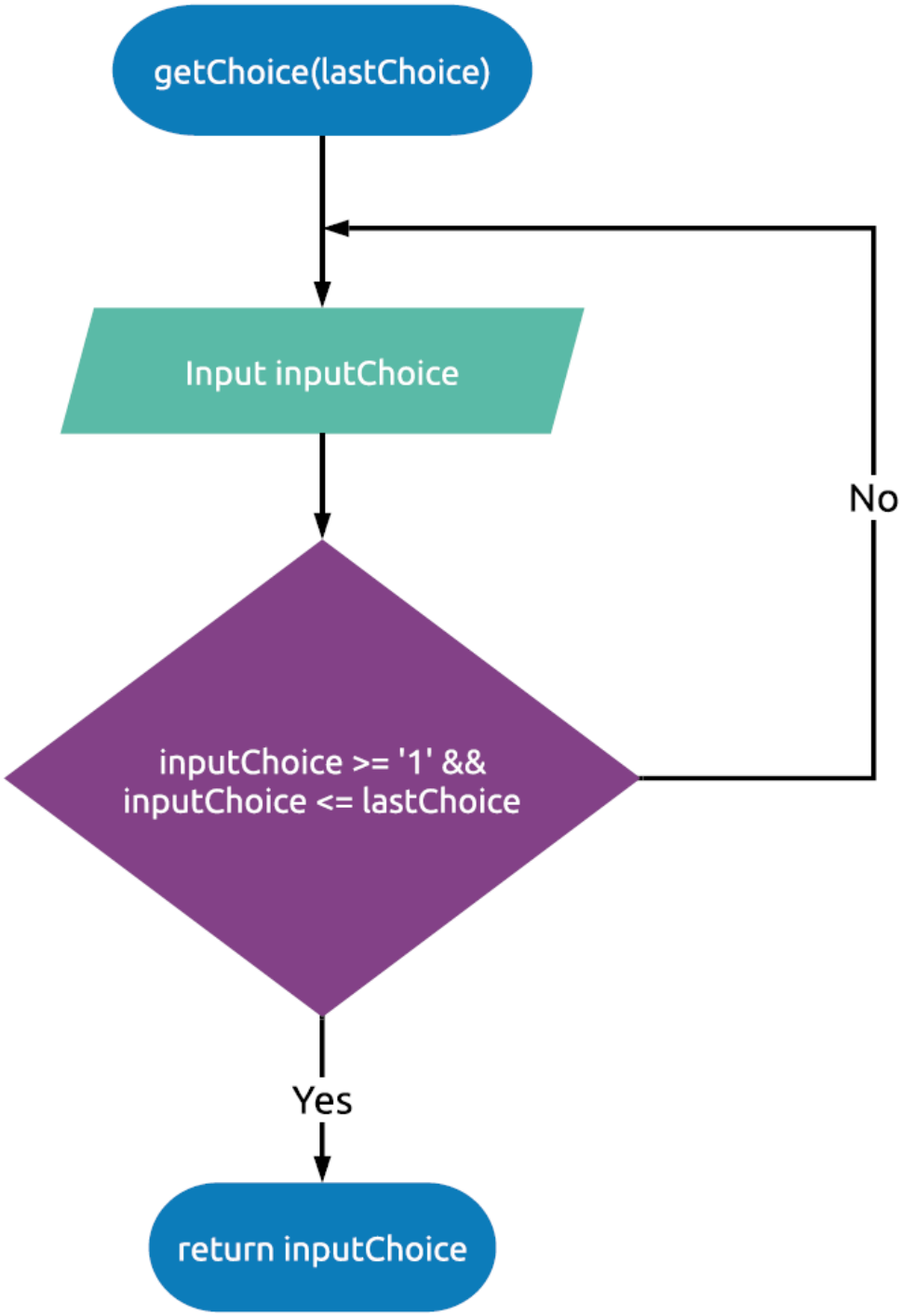
- `bool valid_day(unsigned int day, unsigned int month_end)`



- `bool leap_year(unsigned int year)`



- `char get_choice(char last_choice)`



3. Process Model (Functions)

- `char get_choice(char last_choice)`
  - @param `char last_choice` the last choice that the user can enter
  - @return `char` - a valid response within the possible offered range

  It takes the user choice and checks whether it is in the valid range or not, if it is not, it asks the user for a valid response.

- `bool leap_year(unsigned int year)`
  - @param `unsigned int year` - the year to check whether it's leap or not
  - @return `bool` - returns whether the year is leap or not

  Checks whether the year is leap or not.

- `bool valid_license(unsigned int license)`
  - @param `unsigned int license` - user input license
  - @return returns `bool` - whether the license exist or not

  Checks whether the license is a valid one or not.

- `bool valid_car_plate(unsigned int car_plate)`
  - @param `unsigned int` car_plate - user input car plate
  - @return `bool` - whether the car plate exist or not

  Checks whether the car plate is a valid one or not.

- `bool valid_day(unsigned int day, unsigned int month_end)`
    - @param `unsigned int day` - user input day
    - @param `unsigned int month_end` - user input month total days
    - @return returns `bool` - whether the day is within the month range or not

    Checks whether the day is a valid one or not.

- `bool valid_date(date input_date)`
    - @param `date input_date` - user input date
    - @return returns `bool` - whether the date is valid or not

    Checks whether the date is a valid one or not.

- `bool car_search_name(string owner_name)`
    - @param `string owner_name` - user input owner_name
    - @return returns `bool` - whether the owner name exists or not

    Checks that whether the owner name is a valid one or not, if it is valid, it shows their car/s info then return true, if it is not, it tells the user that is not existed and return false.

- `void car_search(unsigned int car_plate)`
    - @param `unsigned int car_plate` - user input car_plate

    Checks that whether the car plate is a valid one or not, if it is valid, it shows the car info then return true, if it is not, it tells the user that is not existed and return false.

- `void record_fine()`

  It **takes** the car_plate, car_speed, the street_speed_limit, the street name and the fine date from the officer and validate each one of them then calculates the fine value and store it in the car array of recorded fines.

- `void pay_fine()`

  It **takes** the license number from the user, validate it, then checks whether they own cars or not, if they do, it **shows** them the cars and their fines if existed then asks the user which fine he wants to pay for which car and update its status is paid.

- `void show_total_fines()`

  It **takes** the license number from the user; validate it, then **shows** the total amount of unpaid fines for their cars.

- `void register_new_driver()`

  It **takes** the user name, birth date and validate it then it stores his data in the system and **gives** him a license number.

- `void register_new_car()`

  It **takes** for the user license number, validate it, then checks whether he exceeded the maximum number of allowed cars, if he did, it tells them that they can't register a new car, if he didn't, it asks them for the new car info and validate it and **give** it a car plate.

- `void driver_menu()`

  It **shows** the driver them operator and let them **choose** one of them to go to, it validate his choice, if it is valid he go to do his operation.

- `void car_search_menu()`

  It **shows** the search options and let the user **choose** one, it validate their choice if their choice is valid they can search by the way they have chosen.

- `bool main_menu()`

  - @return returns `bool` - whether user want to exit the program or not

  It shows the user his options then validate their choice, if it is valid, a certain function is called and a value returned

  ---

- `int main()`

  It contains an infinite loop that keep calling `main_menu()` every time it returns a value, and if the value is equal to `true`, the program finishes executing, otherwise, it calls `main_menu()` one more time.

4. Coding Style

- ### Naming conventions:

  Using **snake_case**-naming convention with the system variables and functions.

  Snake_case naming convention states that Names (except as noted below) should be all lowercase, with words separated by underscores.

  Example:

  - `unsigned int car_plate`
  - `void car_search_menu()`

  Using **SCREAMING_SNAKE_CASE** for constants.

  SCREAMING_SNAKE_CASE states that constants names should be all uppercase, with words separated by underscores.

  Example:

  - `MAX_OWNED_CARS`

- ### Clean code guidelines:

  - Using meaningful names for the variables that illustrate what data they hold.

  Examples:

    - `char input_choice`
    - `float total_unpaid_fines`

- Using curly braces even when writing one line after the condition or the loop.

Example:

```
if (day > 0 && day <= month_end)
{
     return true;
}
else
{
     return false;
}
```

---

5. Implementation

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

#define MAX_OWNED_CARS 3
#define MIN_MAX_SPEED 40 // in (km/h)
#define MAX_FINES_NUMBER 100
#define MAX_DRIVER_CAPACITY 3000
#define MAX_CARS_NUMBER MAX_OWNED_CARS * MAX_DRIVER_CAPACITY

// current license iterator -> tells how many registered driver
in the system
// used to index the new registered driver

unsigned int current_license;


// constructing a date datatype to hold the day, month and year
// using short int as the big range of the int data type is not
needed
// and unsigned as none of the members ( day, month nor year )
can be negative

struct date
{
     unsigned short int day = 0,
          month = 0,
          year = 0;
};

// constructing a model datatype to hold the car model which
consists of the
// the manufacturer name and the model name
// using string to store both of them as it can hold whatever
characters
// they are consisted of, whether they're numbers or from the
Latin alphabetical

struct model
{
```

```
    string manufacturer = "",
        model_name = "";
};

// constructing a fine datatype to hold the fine information,
its value, the street
// in which it was committed and its date and whether its paid
or not
// using float for the value as it the speed could be a
floating point value so the
// fine value can be as well
// using string for the street name as it can hold a whole
sentence representing the street name
// using the date datatype to store the date of the fine
// using a bool type to store whether the fine is paid or not

struct fine
{
    float value = 0.0;
    string street_name = "";
    date fine_date;
    bool paid = false;
};

// constructing a car datatype to hold the car information, the
owner of the car name,
// the car plate, the production year, the total amount of
fines in the car history, and
// an array to hold and keep track of these fines, and model
datatype to hold the car model
// and the current amount of unpaid fines.
// using string for the owner name as it can hold their whole
name
// using unsigned int for the car plate, production year,
current fine number and the current
// amount of unpaid fines as they all can not be negative and
they fit right in the integer datatype range.
// using array of fine datatype to store the car fines, and a
model datatype to store the car model

struct car
{
```

```
    string car_owner = "";
    unsigned int car_plate = 0, production_year = 0,
        current_fine_number = 0, car_unpaid_fines_number = 0;
    fine recoreded_fines[MAX_FINES_NUMBER];
    model car_model;
}stored_cars[MAX_CARS_NUMBER];

// constructing a driver datatype to hold the driver
information, their name,
// their license number, their birth date, the number of owned
cars
// using unsigned int for the license number and the number of
owned cars neither of
// both can be negative and they fit right in the integer
datatype range
// using string for the owner name as it can hold their whole
name
// using the date datatype to store the driver's birth date

struct driver
{
    unsigned int license_number = 0,
        owned_cars_number = 0;
    string name = "";
    date birth_date;
}stored_drivers[MAX_DRIVER_CAPACITY];

// @param char last_choice the last choice that the user can
enter
// @return returns char - a valid response within the possible
offered range
// takes the user choice and checks whether it is in the valid
range or
// not, if it is not, it asks the user for a valid response.

char get_choice(char);

// @param unsigned int year - the year to check whether it's
leap or not
// @return bool - returns whether the year is leap or not
// it checks whether the year is leap or not
```

```
bool leap_year(unsigned int);

// @param unsigned int license - user input license
// @return returns bool - whether the license exist or not
// checks whether the license is a valid one or not

bool valid_license(unsigned int);

// @param unsigned int car_plate - user input car_plate
// @return returns bool - whether the car plate exist or not
// checks whether the car plate is a valid one or not

bool valid_car_plate(unsigned int);

// @param unsigned int day - user input day
// @param unsigned int month_end - user input month total days
// @return returns bool - whether the day is within the month
range or not
// checks whether the day is a valid one or not

bool valid_day(unsigned int, unsigned int);

// @param date - user input date
// @return returns bool - whether the date is valid or not
// checks whether the date is a valid one or not

bool valid_date(date);

// @param string - user input owner_name
// @return returns bool - whether the owner name exists or not
// checks whether the owner name is a valid one or not, if
// it is valid, it shows his car/s info then return true, if it
is not,
// it tells the user that is not existed and return false

bool car_search_name(string);

// @param unsigned int - user input car_plate
// @return returns bool - whether the car plate exists or not
// checks whether the car palte is a valid one or not, if
// it is valid, it shows the car info then return true, if it
is not,
```

```
// it tells the user that is not existed and return false

void car_search(unsigned int);

// it takes the car_plate, car_speed, the street_speed_limit,
the street name
// and the fine date from the officer and validate each one of
them then
// calculates the fine value and store it in the car array of
recoreded fines

void record_fine();

// it takes the license number from the user, validate it, then
checks whether
// they own cars or not, if they do, it shows them the cars and
their fines if existed
// then asks the user which fine he wants to pay for which car
and update its status is paid

void pay_fine();

// it takes the license number from the user, validate it, then
shows the total
// amount of unpaid fines for their cars

void show_total_fines();

// it takes the user name, birth date and validate it
// then it stores his data in the system and gives him a
license number

void register_new_driver();

// it asks for the user license number, validate it, then
checks whether he
// exceeded the maxiumum number of allowed cars, if he did, it
tells them that
// they can't register a new car, if he didn't, it asks them
for the new car
// info and validate it and give it a car plate
```

```
void register_new_car();

// it shows the driver them operator and let them choose one of
them
// to go to, it validate his choice, if it's valid he go to do
his operation

void driver_menu();

// it shows the search options and let the user choose one, it
validate their choice
// if their choice is valid they can search by the way they've
chosen

void car_search_menu();

// @return returns bool - whether user want to exit the program
or not
// it shows the user his options then validate their choice, if
it's valid
// a certain function gets called and a value returned

bool main_menu();

int main()
{
    while (true)
    {
        if (main_menu() == true)
        {
            system("cls");
            break;
        }
    }
    return 0;
}

char get_choice(char last_choice)
{
    char input_choice;
    while (1)
    {
```

```
        cin >> input_choice;
        if (input_choice >= '1' && input_choice <=
last_choice)
        {
            return input_choice;
        }
        else
        {
            cout << "\n\tPlease Enter a valid response: ";
        }
    }
}

bool leap_year(unsigned int year)
{
    if (year % 4 == 0)
    {
        if (year % 100 == 0)
        {
            if (year % 400 == 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return true;
        }
    }
    else
    {
        return false;
    }
}

bool valid_license(unsigned int license)
{
```

```
    // stored_drivers indexing is based in the driver's
license number
    // for instance, a driver with license 3 will have their
data stored at index 2
    // if the license is set to its intial value which ZERO
then it's not valid
    // if the license number is equal to ZERO then it's not
valid
    if (license > 0 && stored_drivers[license -
1].license_number == 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}

bool valid_car_plate(unsigned int car_plate)
{
    // stored_cars indexing is based in the car's plate number
    // for instance, a car with a plate number 3 will have its
data stored at index 2
    // if the value stored in the car index is euqal to the
value inputed by the user
    // then the car plate is valid
    // if the car plate number is equal to ZERO then it's not
valid
    if (car_plate > 0 && stored_cars[car_plate - 1].car_plate
== car_plate)
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool valid_day(unsigned int day, unsigned int month_end)
{
```

```
    if (day > 0 && day <= month_end)
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool valid_date(date input_date)
{
    if (input_date.year > 0)
    {
        if (input_date.month > 0 && input_date.month < 13)
        {
            switch (input_date.month)
            {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                return valid_day(input_date.day, 31);
                break;
            case 2:
                if (leap_year(input_date.year))
                {
                    return valid_day(input_date.day, 29);
                }
                else
                {
                    return valid_day(input_date.day, 28);
                }
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                return valid_day(input_date.day, 30);
```

```
                    break;
                }
            }
            else
            {
                return false;
            }
        }
        return false;
}

bool car_search_name(string owner_name)
{
        unsigned int name_shows = 0;
        for (unsigned int i = 0; i < MAX_DRIVER_CAPACITY; i++)
        {
            if (owner_name == stored_drivers[i].name)
            {
                name_shows++;
                cout << "\n\t\t" << stored_drivers[i].name;
                if (stored_drivers[i].owned_cars_number == 0)
                {
                    cout << " owns no cars!\n\n\t";
                    system("pause");
                }
                else if (stored_drivers[i].owned_cars_number ==
1)
                {
                    cout << " owns one car\n";
                    unsigned int car_plate = i * 3;
                    cout << "\n\n\tmodel: " <<
stored_cars[car_plate].car_model.manufacturer
                        << ' ' <<
stored_cars[car_plate].car_model.model_name;
                    cout << "\n\n\tproduction year: " <<
stored_cars[car_plate].production_year << "\n\n\t";
                    system("pause");
                }
                else
                {
                    cout << " owns " <<
stored_drivers[i].owned_cars_number << " cars\n";
```

```
                    for (unsigned int car_iterator = 0;
car_iterator < stored_drivers[i].owned_cars_number;
car_iterator++)
                    {
                        unsigned int car_plate = ((i * 3)) +
car_iterator;
                        cout << "\n\n\tmodel: " <<
stored_cars[car_plate].car_model.manufacturer
                            << ' ' <<
stored_cars[car_plate].car_model.model_name;
                        cout << "\n\n\tproduction year: " <<
stored_cars[car_plate].production_year << "\n\n";
                        if (car_iterator + 1 ==
stored_drivers[i].owned_cars_number)
                        {
                            cout << "\n\t";
                            system("pause");
                        }
                    }
                }
            }
        }
        if (name_shows > 0)
        {
            return true;
        }
        return false;
}

void car_search(unsigned int car_plate)
{
    cout << "\n\t" << stored_cars[car_plate - 1].car_owner <<
" owns this car";
    cout << "\n\n\tmodel: " << stored_cars[car_plate -
1].car_model.manufacturer
        << ' ' << stored_cars[car_plate -
1].car_model.model_name;
    cout << "\n\n\tproduction year: " << stored_cars[car_plate
- 1].production_year << "\n\n\t";
    system("pause");
}
```

```
void record_fine()
{
	system("cls");
	cout << "\n\n\tHello Officer!\n";
	unsigned int car_plate;
	float car_speed, road_max_speed;
	date fine_date;
	string street_name;

	cout << "\n\n\tEnter car plate: ";
	cin >> car_plate;

	while (!valid_car_plate(car_plate))
	{
		cout << "\n\n\tcar plate does not exist\n\n";
		cout << "\t1- Enter another car plate\n\n";
		cout << "\t2- Main Menu\n\n\n";
		cout << "\tEnter your choice: ";
		char choice = get_choice('2');
		if (choice == '1')
		{
			cout << "\n\n\tPlease enter a valid car plate: ";
			cin >> car_plate;
		}
		else
		{
			return;
		}
	}

	cout << "\n\n\tEnter road maximum speed (km/h): ";
	cin >> road_max_speed;

	while (road_max_speed < MIN_MAX_SPEED)
	{
		cout << "\n\n\tPlease enter a valid road maximum
speed (km/h): ";
		cin >> road_max_speed;
	}

	cout << "\n\n\tEnter car speed (km/h): ";
	cin >> car_speed;
```

```
    while (car_speed <= road_max_speed)
    {
        cout << "\n\n\tPlease enter a valid exceeding speed
(km/h): ";
        cin >> car_speed;
    }

    cout << "\n\n\tEnter street name: ";
    cin.ignore();
    getline(cin, street_name);
    cout << "\n\n\tEnter the date of the fine\n\t(day - month
- year) space separated: ";
    cin >> fine_date.day >> fine_date.month >> fine_date.year;

    while (!valid_date(fine_date))
    {
        cout << "\n\n\tPlease enter a valid date\n\t(day -
month - year) space separated: ";
        cin >> fine_date.day >> fine_date.month >>
fine_date.year;
    }

    unsigned int fine_number = stored_cars[car_plate -
1].current_fine_number;
    stored_cars[car_plate -
1].recoreded_fines[fine_number].fine_date = fine_date;
    stored_cars[car_plate -
1].recoreded_fines[fine_number].street_name = street_name;
    stored_cars[car_plate -
1].recoreded_fines[fine_number].value = 2 * (car_speed -
road_max_speed);
    stored_cars[car_plate - 1].current_fine_number++;
    stored_cars[car_plate - 1].car_unpaid_fines_number++;

    cout << "\n\n\t";
    system("pause");
}

void pay_fine()
{
    system("cls");
```

```
    unsigned int license_number;
    cout << "\n\n\tEnter your license number: ";
    cin >> license_number;

    while (!valid_license(license_number))
    {
        cout << "\n\n\tlicense number does not exist\n\n";
        cout << "\t1- Enter another car plate\n\n";
        cout << "\t2- Main Menu\n\n\n";
        cout << "\tEnter your choice: ";
        char choice = get_choice('2');
        if (choice == '1')
        {
            cout << "\n\n\tPlease enter a valid license
plate: ";
            cin >> license_number;
        }
        else
        {
            return;
        }
    }

    cout << "\n\n\tYou have";

    if (stored_drivers[license_number - 1].owned_cars_number
== 0)
    {
        cout << " no cars, and no fines!\n\n\t";
        system("pause");
    }

    else if (stored_drivers[license_number -
1].owned_cars_number == 1)
    {
        cout << " one car";
        unsigned int car_index = ((license_number - 1) * 3);
        cout << "\n\n\t\tmodel: " <<
stored_cars[car_index].car_model.manufacturer
            << ' ' <<
stored_cars[car_index].car_model.model_name;
```

```
        cout << "\n\n\t\tproduction year: " <<
stored_cars[car_index].production_year;

        if (stored_cars[car_index].car_unpaid_fines_number >
0)
        {
            cout << "\n\n\t\tThis car has fines: ";
            unsigned int unpaid_fines_number = 0;
            for (unsigned int fine_iterator = 0;
fine_iterator < stored_cars[car_index].current_fine_number;
fine_iterator++)
            {
                if
(stored_cars[car_index].recoreded_fines[fine_iterator].value !=
0.0)
                {
                    if
(stored_cars[car_index].recoreded_fines[fine_iterator].paid ==
false)
                    {
                        unpaid_fines_number++;
                        cout << "\n\n\t\t" <<
unpaid_fines_number << "- Date: ";
                        cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.day << ' ';
                        cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.month << ' ';
                        cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.year;
                        cout << "\n\n\t\t   Street: " <<
stored_cars[car_index].recoreded_fines[fine_iterator].street_na
me;
                        cout << "\n\n\t\t   Value: " <<
stored_cars[car_index].recoreded_fines[fine_iterator].value;
                    }
                    cout << "\n\n";
                }
            }
```

```
                cout << "\n\tChoose the number of fine you want
to pay: ";
                char input_choice = get_choice(char('0' +
unpaid_fines_number));
                unsigned int chosen_fine_number =
int(input_choice) - int('1'), unpaid_fines_iterator = 0;

                for (unsigned int fine_iterator = 0;
fine_iterator < stored_cars[car_index].current_fine_number;
fine_iterator++)
                {
                        if
(stored_cars[car_index].recoreded_fines[fine_iterator].value !=
0.0)
                        {
                                if
(stored_cars[car_index].recoreded_fines[fine_iterator].paid ==
false)
                                {
                                        if (chosen_fine_number ==
unpaid_fines_iterator)
                                        {

    stored_cars[car_index].recoreded_fines[fine_iterator].paid
= true;

    stored_cars[car_index].car_unpaid_fines_number--;
                                                break;
                                        }
                                        unpaid_fines_iterator++;
                                }
                        }
                }

                cout << "\n\n\tFine number " <<
chosen_fine_number + 1 << " is paid\n";
        }
        else if (stored_cars[car_index].current_fine_number
== 0)
        {
                cout << "\n\n\tThis car has no fines at all!";
        }
```

```
        else
        {
            cout << "\n\n\tThis car has no unpaid fines!";
        }

        cout << "\n\n\t";
        system("pause");
    }
    else
    {
        cout << ' ' << stored_drivers[license_number -
1].owned_cars_number << " cars";
        unsigned int
unpaid_fines_number_total[MAX_OWNED_CARS];
        for (int fine_iterator = 0; fine_iterator <
MAX_OWNED_CARS; fine_iterator++)
        {
            unpaid_fines_number_total[fine_iterator] = 0;
        }
        for (unsigned int car_iterator = 0; car_iterator <
stored_drivers[license_number - 1].owned_cars_number;
car_iterator++)
        {
            unsigned int car_index = ((license_number - 1) *
3) + car_iterator;
            cout << "\n\n\t" << car_iterator + 1 << "- model:
"
                <<
stored_cars[car_index].car_model.manufacturer
                << ' ' <<
stored_cars[car_index].car_model.model_name;
            cout << "\n\n\t   production year: " <<
stored_cars[car_index].production_year;
            unsigned int unpaid_fines_number = 0;
            if
(stored_cars[car_index].car_unpaid_fines_number > 0)
            {
                cout << "\n\n\t\tThis car has fines: ";
                for (unsigned int fine_iterator = 0;
fine_iterator < stored_cars[car_index].current_fine_number;
fine_iterator++)
                {
```

```
                        if
(stored_cars[car_index].recoreded_fines[fine_iterator].value !=
0.0)
                        {
                            if
(stored_cars[car_index].recoreded_fines[fine_iterator].paid ==
false)
                            {
                                unpaid_fines_number++;
                                cout << "\n\n\t\t" <<
unpaid_fines_number << "-\n\n\t\tDate: ";
                                cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.day << ' ';
                                cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.month << ' ';
                                cout <<
stored_cars[car_index].recoreded_fines[fine_iterator].fine_date
.year;
                                cout << "\n\n\t\tStreet: " <<
stored_cars[car_index].recoreded_fines[fine_iterator].street_na
me;
                                cout << "\n\n\t\tValue: " <<
stored_cars[car_index].recoreded_fines[fine_iterator].value;
                            }
                            cout << "\n\n";
                        }
                    }
                }
                else if
(stored_cars[car_index].current_fine_number == 0)
                {
                    cout << "\n\n\tThis car has no unpaid
fines!";
                }
                else
                {
                    cout << "\n\n\tThis car has no unpaid
fines!";
                }
```

```
                unpaid_fines_number_total[car_iterator] =
unpaid_fines_number;
            }

            cout << "\n\tChoose the car you want to pay its fine:
";
            char car_choice = get_choice(char('0' +
stored_drivers[license_number - 1].owned_cars_number));
            unsigned int chosen_car_number = (int(car_choice) -
int('1')),
                chosen_car_index = (((license_number - 1) * 3) +
(int(car_choice) - int('1')));

            if
(stored_cars[chosen_car_index].car_unpaid_fines_number == 0)
            {
                cout << "\n\n\tThis car has no unpaid fines!";
                cout << "\n\n\t";
                system("pause");
                return;
            }

            cout << "\n\tChoose the number of fine you want to
pay: ";
            char fine_choice = get_choice(char('0' +
unpaid_fines_number_total[chosen_car_number]));
            unsigned int chosen_fine_number = int(fine_choice) -
int('1'), unpaid_fines_iterator = 0;

            for (unsigned int fine_iterator = 0; fine_iterator <
stored_cars[chosen_car_index].current_fine_number;
fine_iterator++)
            {
                if
(stored_cars[chosen_car_index].recoreded_fines[fine_iterator].v
alue != 0.0)
                {
                    if
(stored_cars[chosen_car_index].recoreded_fines[fine_iterator].p
aid == false)
                    {
```

```
                        if (chosen_fine_number ==
unpaid_fines_iterator)
                        {

     stored_cars[chosen_car_index].recoreded_fines[fine_iterato
r].paid = true;

     stored_cars[chosen_car_index].car_unpaid_fines_number--;
                             break;
                        }
                        unpaid_fines_iterator++;
                    }
               }
          }
          cout << "\n\n\tFine number " << chosen_fine_number +
1 << " is paid\n";
          cout << "\n\n\t";
          system("pause");
     }
}

void show_total_fines()
{
     system("cls");
     unsigned int license_number;
     float total_unpaid_fines = 0.0;
     cout << "\n\n\tEnter your license number: ";
     cin >> license_number;
     while (!valid_license(license_number))
     {
          cout << "\n\n\tlicense number does not exist\n\n";
          cout << "\t1- Enter another car plate\n\n";
          cout << "\t2- Main Menu\n\n\n";
          cout << "\tEnter your choice: ";
          char choice = get_choice('2');
          if (choice == '1')
          {
               cout << "\n\n\tPlease enter a valid license
plate: ";
               cin >> license_number;
          }
          else
```

```
        {
            return;
        }
    }

    for (unsigned int car_iterator = 0; car_iterator <
MAX_OWNED_CARS; car_iterator++)
    {
        for (unsigned int fine_iterator = 0; fine_iterator <
MAX_FINES_NUMBER; fine_iterator++)
        {
            unsigned int car_index = (((license_number - 1) *
3) + car_iterator);
            if
(stored_cars[car_index].recoreded_fines[fine_iterator].paid ==
false)
            {
                total_unpaid_fines +=
stored_cars[car_index].recoreded_fines[fine_iterator].value;
            }
        }
    }

    cout << "\n\n\tThe total amount of unpaid fines for your
cars: ";
    cout << total_unpaid_fines << " \n\n\t";
    system("pause");
}

void register_new_driver()
{
    system("cls");
    string name;
    date birth_date;
    cout << "\n\n\tEnter your name: ";
    cin.ignore();
    getline(cin, name);
    cout << "\n\tEnter your birth date\n\n\t (day - month -
year) space separated: ";
    cin >> birth_date.day >> birth_date.month >>
birth_date.year;
    while (!valid_date(birth_date))
```

```
    {
        cout << "\n\n\tPlease enter a valid birth date!\n";
        cout << "\n\tEnter your birth date\n\n\t (day - month
- year) space separated: ";
        cin >> birth_date.day >> birth_date.month >>
birth_date.year;
    }
    stored_drivers[current_license].license_number =
current_license + 1;
    stored_drivers[current_license].birth_date = birth_date;
    stored_drivers[current_license].name = name;
    current_license++;

    cout << "\n\n\tWelcome to the traffic contorl system " <<
name << " !\n\n";
    cout << "\tYour license number: " << current_license << "
\n\n";
    cout << '\t';
    system("pause");
}

void register_new_car()
{
    system("cls");
    unsigned int license_number;
    cout << "\n\n\tEnter your license number: ";
    cin >> license_number;

    while (!valid_license(license_number))
    {
        cout << "\n\n\tlicense number does not exist\n\n";
        cout << "\t1- Enter another car plate\n\n";
        cout << "\t2- Main Menu\n\n\n";
        cout << "\tEnter your choice: ";
        char choice = get_choice('2');
        if (choice == '1')
        {
            cout << "\n\n\tPlease enter a valid license
plate: ";
            cin >> license_number;
        }
        else
```

```
        {
            return;
        }
    }

    if (stored_drivers[license_number - 1].owned_cars_number
== MAX_OWNED_CARS)
    {
        cout << "\n\n\tYou have reached the maxiumum number
of owned cars!";
        cout << "\n\n\tYou can not register a new car!";
        cout << "\n\n\t";
        system("pause");
        return;
    }

    car new_car;
    cout << "\n\tEnter the manufacturer of the car: ";
    cin >> new_car.car_model.manufacturer;
    cout << "\n\tEnter the model of the car: ";
    cin >> new_car.car_model.model_name;
    cout << "\n\tEnter the production year of the car: ";
    cin >> new_car.production_year;
    while (new_car.production_year < 0)
    {
        cout << "\n\n\tEnter a valid production year: ";
        cin >> new_car.production_year;
    }
    new_car.car_owner = stored_drivers[license_number -
1].name;
    new_car.car_plate = (((license_number - 1) * 3) +
stored_drivers[license_number - 1].owned_cars_number) + 1;
    stored_cars[new_car.car_plate - 1] = new_car;
    stored_drivers[license_number - 1].owned_cars_number++;

    cout << "\n\n\tWelcome back " <<
stored_drivers[license_number - 1].name << " !\n\n";
    cout << "\tYour new registered car plate: " <<
new_car.car_plate << " \n\n";
    cout << "\t\t\t\tmodel: " <<
new_car.car_model.manufacturer <<
        ' ' << new_car.car_model.model_name << " \n\n";
```

```
    cout << "\t\t\t\tproduction year: " <<
new_car.production_year << " \n\n";
    cout << '\t';
    system("pause");
}

void driver_menu()
{
    system("cls");
    cout << "\n\n\tHello Driver!\n\n";
    cout << "\t1- Register new driver\n\n";
    cout << "\t2- Register new car\n\n";
    cout << "\t3- Show total unpaid fines\n\n";
    cout << "\t4- Pay fine\n\n\n";
    cout << "\tPlease choose your operation: ";
    char choice = get_choice('4');

    if (choice == '1')
    {
        register_new_driver();
    }
    else if (choice == '2')
    {
        register_new_car();
    }
    else if (choice == '3')
    {
        show_total_fines();
    }
    else if (choice == '4')
    {
        pay_fine();
    }
}

void car_search_menu()
{
    system("cls");
    cout << "\n\n\tHello!, How would you like to search?\n\n";
    cout << "\t1- By owner's name\n\n";
    cout << "\t2- By car plate\n\n";
    cout << "\n\tChoose (1 or 2): ";
```

```
    char choice = get_choice('2');

    if (choice == '1')
    {
        system("cls");
        string name;
        cout << "\n\n\tEnter owner's name: ";
        cin.ignore();
        getline(cin, name);
        while (!car_search_name(name))
        {
            cout << "\n\tThere is no such a car owner!\n\n";
            cout << "\t1- Enter another car owner name\n\n";
            cout << "\t2- Main Menu\n\n\n";
            cout << "\tEnter your choice: ";
            char choice = get_choice('2');
            if (choice == '1')
            {
                cout << "\n\n\tPlease enter an existing car
owner name: ";
                getline(cin, name);;
            }
            else
            {
                return;
            }
        }
    }
    else if (choice == '2')
    {
        system("cls");
        unsigned int car_plate;
        cout << "\n\n\tEnter car plate: ";
        cin >> car_plate;
        while (!valid_car_plate(car_plate))
        {
            cout << "\n\n\tcar plate does not exist\n\n";
            cout << "\t1- Enter another car plate\n\n";
            cout << "\t2- Main Menu\n\n\n";
            cout << "\tEnter your choice: ";
            char choice = get_choice('2');
            if (choice == '1')
```

```
                {
                    cout << "\n\n\tPlease enter a valid car
plate: ";
                    cin >> car_plate;
                }
                else
                {
                    return;
                }
            }
            car_search(car_plate);
        }
}

bool main_menu()
{
    system("cls");
    cout << "\n\n\t1- Driver Options\n\n";
    cout << "\t2- Record fine (Officer ONLY)\n\n";
    cout << "\t3- Search for a car\n\n";
    cout << "\t4- Exit\n\n\n";
    cout << "\tThis is Traffic Contorl System, Enter your
choice: ";

    char choice = get_choice('4');

    if (choice == '1')
    {
        driver_menu();
        return false;
    }
    else if (choice == '2')
    {
        record_fine();
        return false;
    }
    else if (choice == '3')
    {
        car_search_menu();
        return false;
    }
    else
```

```
    {
        return true;
    }
}
```

6. Testing

- `bool main_menu()`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

    1- Driver Options

    2- Record fine (Officer ONLY)

    3- Search for a car

    4- Exit

    This is Traffic Contorl System, Enter your choice: _
```

- `char get_choice(char last_choice)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

    1- Driver Options

    2- Record fine (Officer ONLY)

    3- Search for a car

    4- Exit

    This is Traffic Contorl System, Enter your choice: 6

    Please Enter a valid response:
```

- void driver_menu()

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Hello Driver!

1- Register new driver

2- Register new car

3- Show total unpaid fines

4- Pay fine

Please choose your operation: _
```

- void register_new_driver()

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your name: Nabil Asaad

Enter your birth date

 (day - month - year) space separated: 18 3 1992

Welcome to the traffic contorl system Nabil Asaad !

Your license number: 1

Press any key to continue . . .
```

- void register_new_car()

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your license number: 1

Enter the manufacturer of the car: Audi

Enter the model of the car: A5

Enter the production year of the car: 2018

Welcome back Nabil Asaad !

Your new registered car plate: 1

                        model: Audi A5

                        production year: 2018

Press any key to continue . . .
```

- `void show_total_fines()`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your license number: 1

The total amount of unpaid fines for your cars: 300

Press any key to continue . . .
```

- `void pay_fine()`.

  1.

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your license number: 1

You have one car

        model: Audi A5

        production year: 2018

        This car has fines:

        1- Date: 19 5 2020

           Street: Central Park street

           Value: 100


        2- Date: 12 3 2020

           Street: Central Zoo street

           Value: 200

Choose the number of fine you want to pay: 2

Fine number 2 is paid

Press any key to continue . . .
```

2.

C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

```
Enter your license number: 1

You have 2 cars
1- model: Audi A5
   production year: 2018
        This car has fines:
        1-
        Date: 19 5 2020
        Street: Central Park street
        Value: 100



2- model: Genesis G70
   production year: 2019
        This car has fines:
        1-
        Date: 23 5 2020
        Street: US 6
        Value: 300

Choose the car you want to pay its fine: 2
Choose the number of fine you want to pay: 1

Fine number 1 is paid

Press any key to continue . . .
```

- `bool valid_car_plate(unsigned int car_plate)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Hello Officer!

Enter car plate: 5

car plate does not exist, Please enter a valid car plate:
```

- `bool valid_license(unsigned int license)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your license number: 5

Enter a valid license number: _
```

- `bool valid_date(date input_date)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter your name: Nabil Asaad
Enter your birth date
 (day - month - year) space separated: 18 13 1992

Please enter a valid birth date!
Enter your birth date
 (day - month - year) space separated:
```

- `void car_search_menu()`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Hello!, How would you like to search?

1- By owner's name

2- By car plate

Choose (1 or 2):
```

- `bool car_search_name(string owner_name)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter owner's name: Nabil Asaad

        Nabil Asaad owns 2 cars

model: Audi A5

production year: 2018


model: Genesis G70

production year: 2018

Press any key to continue . . .
```

- `void car_search(unsigned int car_plate)`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Enter car plate: 2

Nabil Asaad owns this car

model: Genesis G70

production year: 2018

Press any key to continue . . .
```

- `void record_fine()`

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Hello Officer!

Enter car plate: 1

Enter road maximum speed (km/h): 150

Enter car speed (km/h): 300

Enter street name: US 7

Enter the date of the fine
(day - month - year) space separated: 26 5 2020

Press any key to continue . . .
```

- **A rare case**
  - In a function that takes an input from the user to search for it or to store data related to it, such as `record_fine()`, `show_total_fines()`, `car_search(unsigned int car_plate)`, and others.

If the user forgot the correct input or entered an incorrect one or there is not any data yet in the system, the user will be asked whether they want to enter a new one or to go back to the main menu.

```
C:\Users\egypt2\source\repos\TrafficControlSystem\Debug\TrafficControlSystem.exe

Hello Officer!

Enter car plate: 1

car plate does not exist

1- Enter another car plate

2- Main Menu

Enter your choice:
```

References:

[1] Bridger, A., & Pisano, J. (2001). C++ coding standards.

[2] Simison, Graeme. C. & Witt, Graham. C. (2005). Data Modeling Essentials. Third Edition.

[3] Alan Dennis, Barbara Haley Wixom & Roberta M. Roth (2014). System Analysis and Design.

[4] Collopy, D. M. (1998). Introduction to C++ Programming: A Modular Approach.

[5] Busbee, K. L. & Braunschweig, D. (2018). Programming Fundamentals Modular Structured Approach, 2nd Edition.

[6] Sutter, H., & Alexandrescu, A. (2004). C++ coding standards: 101 rules, guidelines, and best practices.

[7] Meyers, S. (2005). Effective C++: 50 specific ways to improve your programs and designs.