PARKING TICKET SIMULATOR

Design Document

Abstract

The Parking Ticket Simulator demonstrates multiple classes collaborating to produce a parking ticket.

Mickie Blair

Table of Contents

Table of Figures

Parking Ticket Simulator

Description:

The program demonstrates multiples classes collaborating to produce a parking ticket issued by a police officer based on the time the car has been parked in comparison to the amount of time paid on the meter. The program calculates the amount of the fine based on the minutes the car has been illegally parked.

Programming Strategy:

In the program, a police officer will be asked to enter a valid name and badge number. Next, the program will ask the police officer for the minutes the car has been parked and the minutes paid at the meter. If the car is illegally parked, the police officer will be asked to enter the make, model, license of the car. Once valid information has been entered, a parking ticket object will be created. The parking ticket object will ask the police officer for the location of the violation and the meter number. The parking ticket object will calculate the fine and display the parking ticket.

Program Classes

The program is broken down into several classes to represent each object. The classes are described below:

Class	**Responsibilities**
ParkedCar	• To know the car's make, model, color, license number, and the number of minutes the car has been parked
ParkingMeter	• To know the number of minutes paid purchased at the meter
ParkingTicket	• To report the make, model, color, and license number of the illegally parked car. • To report the amount of the fine • To report the name and badge number of the officer issuing the ticket
PoliceOfficer	• To know the police officer's name and badge number • To examine a ParkedCar and ParkingMeter to determine if a ticket need to be issued • To issue a ParkingTicket, if the car is illegally parked.
Validation	• To validate the input of the user using static methods.

Table 1 - Program Classes

Program Functions

The program is broken down into several functions to separate different areas of the program. The Program includes the header files Prototypes.h, ParkedCar.h, ParkingMeter.h, ParkingTicket.h, and PoliceOfficer.h.

Function	Description
main	The program's main function. It calls the programs other functions from inside a try block to catch insufficient memory exceptions.
repeatProgram	Asks the user if they would like to run the program again. User input is validated. Calls the showIntro and ParkingTicketDemo Functions .
showIntro	Displays an introduction to the program. Takes no arguments.
parkingTicketDemo	Calls the inputOfficerInfo function
inputOfficerInfo	Contains a loop to ask for officer details (getOfficerDetails) and inspect a car and meter(inspectCarMeter). The loop repeats if a different officer would like to enter information
getOfficerDetails	Asks the officer for name and badge number using overloaded cin
inspectCarMeter	Ask for the minutes parked and the minutes paid inside a loop allowing the same officer to enter multiple cars. Call the officer function to examine the car and meter.

Table 2 - Program Functions

main Function

The main function will call the function to repeat the program if sufficient memory exists. The main function also sets the size of the console window to display the entire parking ticket if generated.

Pseudocode for main

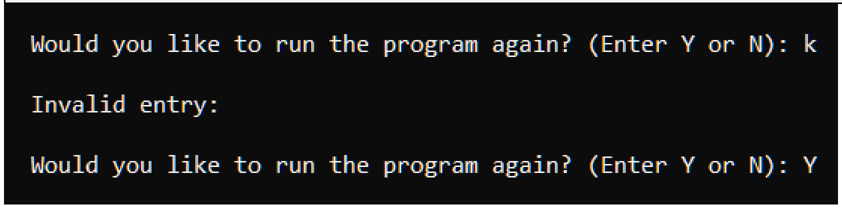
```
set console size

try
       {
              repeating the program

       }

catch (bad_alloc)
       {
              error insufficient memory
       }
```

repeatProgram Function

The function uses a do while loop to ask the user if they would like to run the program again. The user input is validated.   Inside the do while loop the showIntro function and repeatTest functions are called. The only variable is a string to hold the user input to continue or exit. It takes no arguments.

Pseudocode

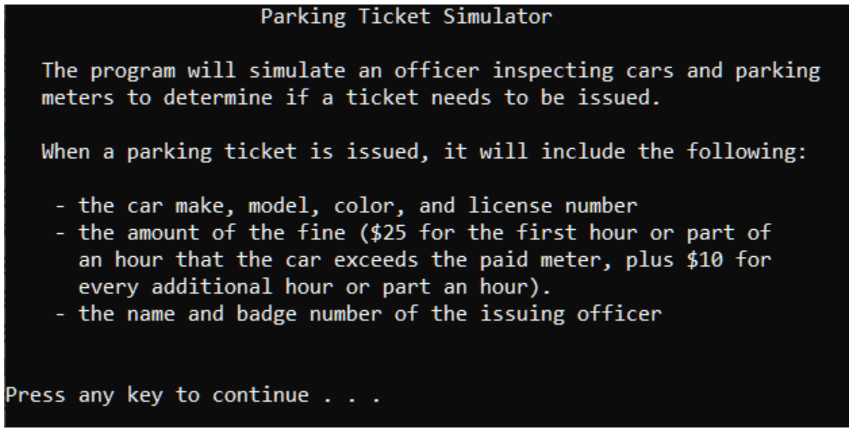
```
do
{
    showIntro

    repeatMenu

    input do again
     {
        validate do again
     }
}
while user want to continue
```



```
Would you like to run the program again? (Enter Y or N): k

Invalid entry:

Would you like to run the program again? (Enter Y or N): Y
```

showIntro Function

The showIntro Function displays a brief description of the program.



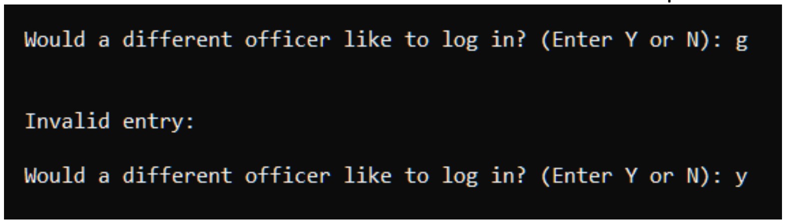
```
                   Parking Ticket Simulator

  The program will simulate an officer inspecting cars and parking
  meters to determine if a ticket needs to be issued.

  When a parking ticket is issued, it will include the following:

   - the car make, model, color, and license number
   - the amount of the fine ($25 for the first hour or part of
     an hour that the car exceeds the paid meter, plus $10 for
     every additional hour or part an hour).
   - the name and badge number of the issuing officer


Press any key to continue . . .
```

ParkingTicketDemo Function

The function clears the screen and calls the inputOfficerInfo function.

inputOfficerInfo Function

The function creates an officer object and then enters a loop to call the function to ask for officer details (getOfficerDetails) and the function inspect a car and meter(inspectCarMeter). The loop repeats to ask if a different officer would like to enter information. Input is validated.



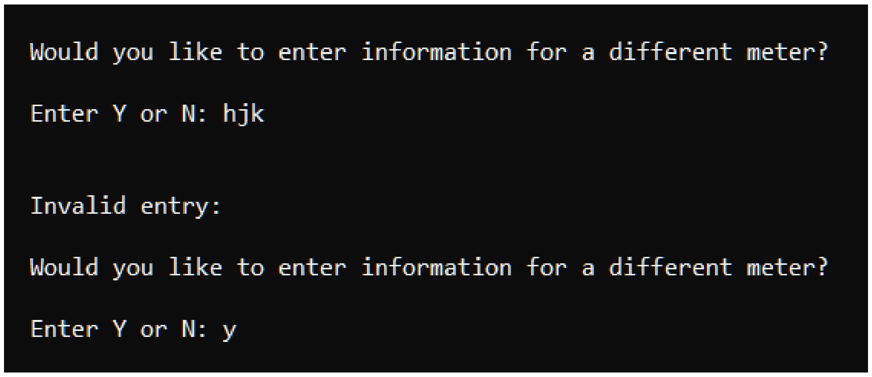
```
Would a different officer like to log in? (Enter Y or N): g


Invalid entry:

Would a different officer like to log in? (Enter Y or N): y
```

getOfficerDetails Function

The function uses an overloaded cin to set the name and badge number of the officer after being validated. The function receives a reference to the PoliceOfficer object created in the inputOfficerInfo function.

inspectCarMeter Function

The function creates a ParkedCar and ParkingMeter Object. Inside a loop, the user sets the minutes the car has been parked in the ParkedCar object. The loop then uses an overloaded cin to set the minutes paid of the ParkingMeter object. Input is validated for the loop.



```
Would you like to enter information for a different meter?

Enter Y or N: hjk


Invalid entry:

Would you like to enter information for a different meter?

Enter Y or N: y
```

ParkedCar Class:

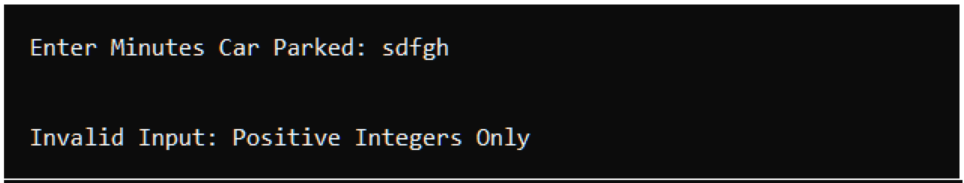
The ParkedCar Class contains a no args constructor and a constructor which takes parameters for all following variables . The setters and getters for each variable is listed.

Variable (private)	Type	Setter (public)	Getter (public)
make	string	`void setMake(string);`	`string getMake();`
model	string	`void setModel(string);`	`string getModel();`
color	string	`void setColor(string);`	`string getColor();`
license	string	`void setLicense(string);`	`string getLicense();`
minParked	int	`void setMinParked();`	`int getMinParked();`

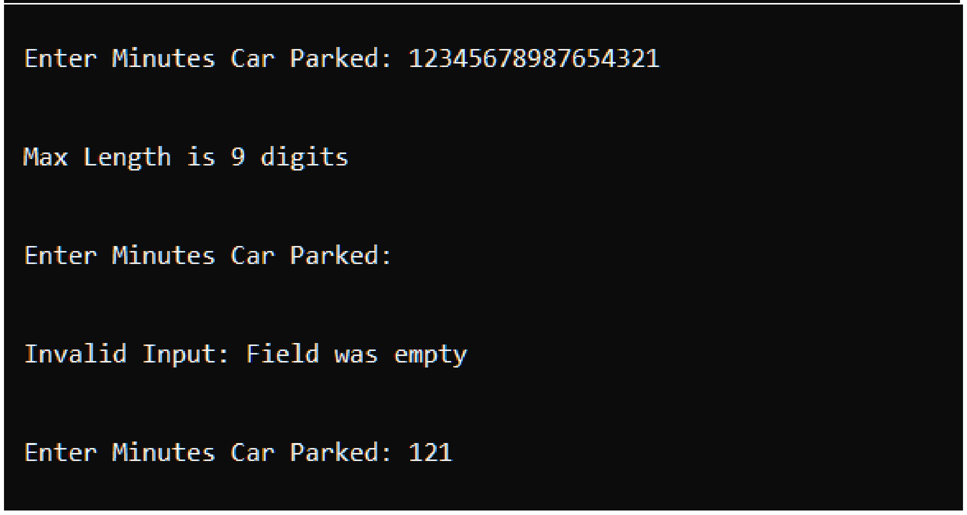
Table 3 - ParkedCar Variables

setMinutesParked Function

The function is called independently of the overloaded cin for the ParkedCar Class to allow the user to skip entering the remaining variables if a ticket is not issued. The setMinParked function uses the Validation Class to ensure the entry is valid.



```
Enter Minutes Car Parked: sdfgh

Invalid Input: Positive Integers Only
```



```
Enter Minutes Car Parked: 12345678987654321

Max Length is 9 digits

Enter Minutes Car Parked:

Invalid Input: Field was empty

Enter Minutes Car Parked: 121
```

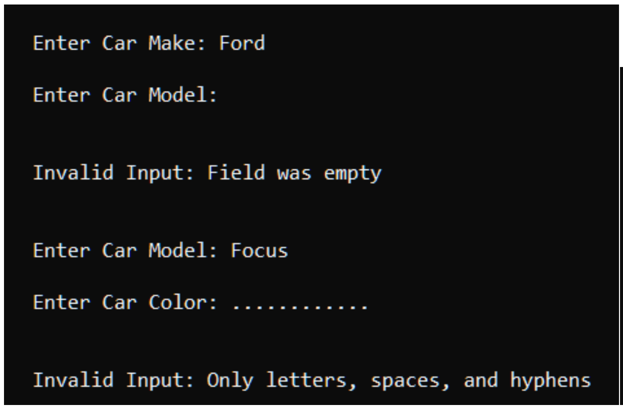
Friend Functions of the ParkedCar Class

operator << Overloaded Function

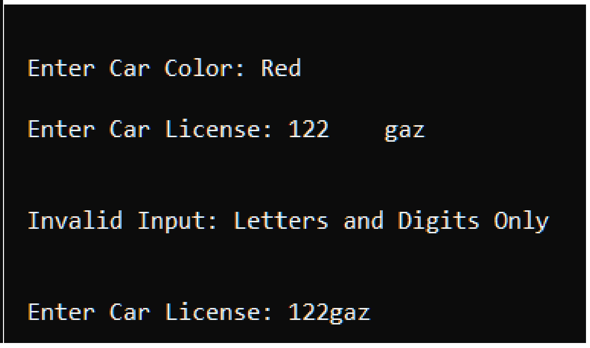
Takes an ostream reference object and constant ParkedCar reference object and returns a reference to the ParkedCar variable in the specified format. This allows cout to display the private member variables values without using the dot operator.

operator >> Overloaded Function

Takes an istream reference object and ParkedCar reference object and returns a reference to make, model, color, and license number of the parked car.   Input is validated using the Validation Class.



```
Enter Car Make: Ford

Enter Car Model:

Invalid Input: Field was empty

Enter Car Model: Focus

Enter Car Color: ...........

Invalid Input: Only letters, spaces, and hyphens
```



```
Enter Car Color: Red

Enter Car License: 122    gaz

Invalid Input: Letters and Digits Only

Enter Car License: 122gaz
```

Class Declaration

The class specification file (ParkedCar.h) contains the declaration statements for the variables and functions that are members are of the class.

```
#ifndef PARKEDCAR_H
#define PARKEDCAR_H

#include <iostream>
#include <string>

using namespace std;

class ParkedCar {

private:
	string make;		//car make
	string model;		//car model
	string color;		//car color
	string license;			//car license
	int minParked;			//minutes parked

public:
	//constructor - no args
	ParkedCar();

	//constructor
	ParkedCar(string, string, string, string, int);

	//setters
	void setMake(string);
	void setModel(string);
	void setColor(string);
	void setLicense(string);
	void setMinParked();

	//getters
	string getMake();
	string getModel();
	string getColor();
	string getLicense();
	int getMinParked();


	//friends;
	friend ostream& operator<<(ostream&, const ParkedCar&);
	friend istream& operator>>(istream&, ParkedCar&);

};

#endif
```
Table 4 - ParkedCar Declaration

Class Implementation

The class implementation file (Parked.cpp) contains the definitions for the public functions of the class.

```
#include "ParkedCar.h"
#include "Validation.h"

#include <iostream>
#include <string>
#include <algorithm>


//constructor - no args;
ParkedCar::ParkedCar()
{
       make = "";
       model = "";
       color = "";
       license = "";
       minParked = 0;
}

//constructor;
ParkedCar::ParkedCar(string cMake, string cModel, string cColor,
                                  string cLisc, int cMinParked)
{
       make = cMake;
       model = cModel;
       color = cColor;
       license = cLisc;
       minParked = cMinParked;
}

//set Make
void ParkedCar::setMake(string cMake)
{
       make = cMake;
}

//set Model
void ParkedCar::setModel(string cModel)
{
       model = cModel;
}

//set Color
void ParkedCar::setColor(string cColor)
{
       color = cColor;
}

//set License
void ParkedCar::setLicense(string cLisc)
{
       license = cLisc;
}


//set minutes parked
void ParkedCar::setMinParked()
```

```
{
	string carMinParked;			//string of minutes parked
	bool validMinParked = false;		//valid min parked
	bool isValid = false;				//is valid boolean

	// do with try catch for validation
	do
	{
		try
		{
			cout << "\n   Enter Minutes Car Parked: ";
			getline(cin, carMinParked);

			if (carMinParked.empty())
			{
				throw Validation::EmptyInput();
			}

			validMinParked = Validation::isInteger(carMinParked);

			if (!validMinParked)
			{
				throw Validation::NumbersOnly();
			}

			minParked = stoi(carMinParked);

			isValid = true;
		}

		catch (Validation::EmptyInput)
		{
			cout << "\n\n   Invalid Input: Field was empty\n" << endl;
		}

		catch (Validation::NumbersOnly)
		{
			cout << "\n\n   Invalid Input: Positive Integers Only\n" << endl;
		}

		catch (out_of_range)
		{
			cout << "\n\n   Max Length is 9 digits\n" << endl;
		}

	} while (!isValid);
}

//get make
string ParkedCar::getMake()
{
	return make;
}




//get model
string ParkedCar::getModel()
```

```cpp
{
	return model;
}

//get color
string ParkedCar::getColor()
{
	return color;
}

//get license
string ParkedCar::getLicense()
{
	return license;
}

//get minutes parked
int ParkedCar::getMinParked()
{
	return minParked;
}


//overloaded cout
ostream& operator<<(ostream& strm, const ParkedCar& obj)
{
	//create strm of officer Name and Badge
	strm << "Make: " << obj.make << endl
		<< "Model: " << obj.model << endl
		<< "Color: " << obj.color << endl
		<< "License: " << obj.license << endl
		<< "Minutes Parked: " << obj.minParked << endl;

	return strm;
}

//overloaded cin
istream& operator>>(istream& strm, ParkedCar& obj)
{
	string carMake;				//car make
	string carModel;			//car model
	string carColor;			//car color
	string carLicense;			//car license

	bool isValid = false;			//bool for is Valid
	bool validCarMake = false;		//bool for valid car make
	bool validCarModel = false;		//bool for valid car model
	bool validCarColor = false;		//bool for valid car color
	bool validCarLicense = false;	//bool for valid car license
	bool validLength = false;		//bool for valid length

	//get car info and validate information
	do
	{



		try
		{
			//set car make if not valid
			do
			{

				if (!validCarMake)
				{
					cout << "\n    Enter Car Make: ";
					getline(cin, carMake);

					if (carMake.empty())
					{
						throw Validation::EmptyInput();
					}

				       validLength = Validation::isValidLength(carMake, 16);

					if (!validLength)
					{
						throw Validation::MaxLength();
					}

					transform(carMake.begin(), carMake.end(),
						  carMake.begin(), toupper);

					validCarMake = Validation::isValidName(carMake);

					if (!validCarMake)
					{
						throw Validation::InvalidInput();
					}

					validCarMake = true;

				}

			} while (!validCarMake);

			obj.setMake(carMake);

			//set car model if not valid
			do
			{

				if (!validCarModel)
				{
					cout << "\n    Enter Car Model: ";
					getline(cin, carModel);

					if (carModel.empty())
					{
						throw Validation::EmptyInput();
					}

				      validLength = Validation::isValidLength(carModel, 16);

					if (!validLength)
					{
						throw Validation::MaxLength();
					}
```

```
                                transform(carModel.begin(), carModel.end(),
                                          carModel.begin(), toupper);

                                validCarModel = Validation::isValidName(carModel);

                                if (!validCarModel)
                                {
                                        throw Validation::InvalidInput();
                                }

                                validCarModel = true;

                        }

                } while (!validCarModel);

                obj.setModel(carModel);

                //set car color if not valid
                do
                {

                        if (!validCarColor)
                        {
                                cout << "\n   Enter Car Color: ";
                                getline(cin, carColor);

                                if (carColor.empty())
                                {
                                        throw Validation::EmptyInput();
                                }

                              validLength = Validation::isValidLength(carColor, 16);

                                if (!validLength)
                                {
                                        throw Validation::MaxLength();
                                }

                                transform(carColor.begin(), carColor.end(),
                                          carColor.begin(), toupper);

                                validCarColor = Validation::isValidName(carColor);

                                if (!validCarColor)
                                {
                                        throw Validation::InvalidInput();
                                }

                                validCarColor = true;

                        }

                } while (!validCarColor);

                obj.setColor(carColor);

                //set car license if not valid
                do
                {
                        if (!validCarLicense)
                        {
                                cout << "\n   Enter Car License: ";
                                getline(cin, carLicense);

                                if (carLicense.empty())
                                {
                                        throw Validation::EmptyInput();
                                }

                            validLength = Validation::isValidLength(carLicense, 16);

                                if (!validLength)
                                {
                                        throw Validation::MaxLength();
                                }

                                transform(carLicense.begin(), carLicense.end(),
                                           carLicense.begin(), toupper);

                                validCarLicense =
                                    Validation::isLettersOrNumbers(carLicense);

                                if (!validCarLicense)
                                {
                                        throw Validation::NumbersOrLetters();
                                }

                                validCarLicense = true;

                        }

                } while (!validCarLicense);

                obj.setLicense(carLicense);

                isValid = true;
        }

        catch (Validation::InvalidInput)
        {
                cout << "\n\n   Invalid Input: Only letters, spaces, and
                          hyphens\n" << endl;
        }

        catch (Validation::EmptyInput)
        {
                cout << "\n\n   Invalid Input: Field was empty\n" << endl;
        }

        catch (Validation::NumbersOnly)
        {
                cout << "\n\n   Invalid Input: Positive Integers Only\n" << endl;
        }

        catch (Validation::NumbersOrLetters)
        {
```

```
                    cout << "\n\n   Invalid Input: Letters and Digits Only\n" << endl;
            }

            catch (out_of_range)
            {
                    cout << "\n\n   Max Length is 9 digits\n" << endl;
            }

            catch (Validation::MaxLength)
            {
                    cout << "\n\n   Invalid Input: Maximum Length of Input is 16
                            characters\n" << endl;
            }


        } while (!isValid);

        return strm;
}
```
Table 5 - ParkedCar Implementation

ParkingMeter Class:

The ParkingMeter Class contains a no args constructor and a constructor which takes parameters for the following variable . The setter and getter for the variable is listed.

Variable (private)	Type	Setter (public)	Getter (public)
minPaid	int	`void setMinPaid(int);`	`int getMinPaid();`

Table 6 - ParkingMeter Variables

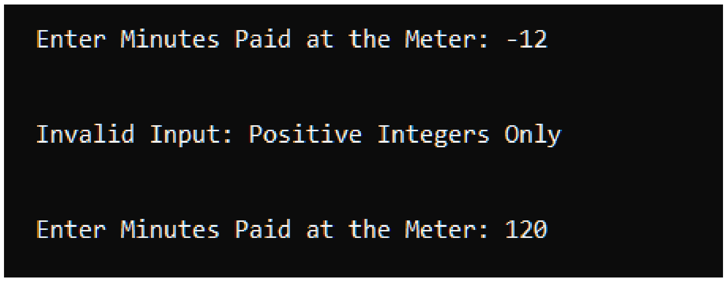
Friend Functions of the ParkedMeter Class

operator << Overloaded Function

Takes an ostream reference object and constant ParkingMeter reference object and returns a reference to the ParkingMeter variable in the specified format. This allows cout to display the private member variables values without using the dot operator.

operator >> Overloaded Function

Takes an istream reference object and ParkingMeter reference object and returns a reference minutes paid. Input is validated using the Validation Class.


```
Enter Minutes Paid at the Meter: -12


Invalid Input: Positive Integers Only


Enter Minutes Paid at the Meter: 120
```

Class Declaration

The class specification file (ParkingMeter.h) contains the declaration statements for the variables and functions that are members are of the class.

```
#ifndef PARKINGMETER_H
#define PARKINGMETER_H

#include <iostream>
#include <string>

using namespace std;

class ParkingMeter {

private:
	int minPaid;

public:
	//constructor - no args
	ParkingMeter();

	//constructor
	ParkingMeter(int);

	//setters
	void setMinPaid(int);

	//getters
	int getMinPaid();

	//friends
	friend ostream& operator<<(ostream&, const ParkingMeter&);
	friend istream& operator>>(istream&, ParkingMeter&);
};

#endif
```

Table 7 - ParkingMeter Declaration

Class Implementation

The class implementation file (ParkingMeter.cpp) contains the definitions for the public functions of the class.

```
#include "ParkingMeter.h"
#include "Validation.h"

//constructor
ParkingMeter::ParkingMeter()
{
	minPaid = 0;
}

//constructor with minutes paid
ParkingMeter::ParkingMeter(int paid)
{
	minPaid = paid;
}
```

```
//set minutes paid
void ParkingMeter::setMinPaid(int paid)
{
	minPaid = paid;
}

//get minutes paid
int ParkingMeter::getMinPaid()
{
	return minPaid;
}

//overload the cout
ostream& operator<<(ostream& strm, const ParkingMeter& obj)
{
	//create strm of officer Name and Badge
	strm << "Minutes Paid: " << obj.minPaid << endl;

	return strm;
}

//overload the cin
istream& operator>>(istream& strm, ParkingMeter& obj)
{
	string meterMinPaid;

	bool isValid = false;
	bool validMinPaid = false;

	//get parking meter info and validate information
	do
	{
		try
		{
			//set minutes parked

			cout << "\n   Enter Minutes Paid at the Meter: ";
			getline(cin, meterMinPaid);

			if (meterMinPaid.empty())
			{
				throw Validation::EmptyInput();
			}

			validMinPaid = Validation::isInteger(meterMinPaid);
			if (!validMinPaid)
			{
				throw Validation::NumbersOnly();
			}

			obj.setMinPaid(stoi(meterMinPaid));

			isValid = true;
		}

		catch (Validation::EmptyInput)
		{
			cout << "\n\n   Invalid Input: Field was empty\n" << endl;
```

```
              }

              catch (Validation::NumbersOnly)
              {
                     cout << "\n\n   Invalid Input: Positive Integers Only\n" << endl;
              }

              catch (out_of_range)
              {
                     cout << "\n\n   Max Length is 9 digits\n" << endl;
              }



       } while (!isValid);

       return strm;
}
```
Table 8 - ParkingMeter Implementation

PoliceOfficer Class:

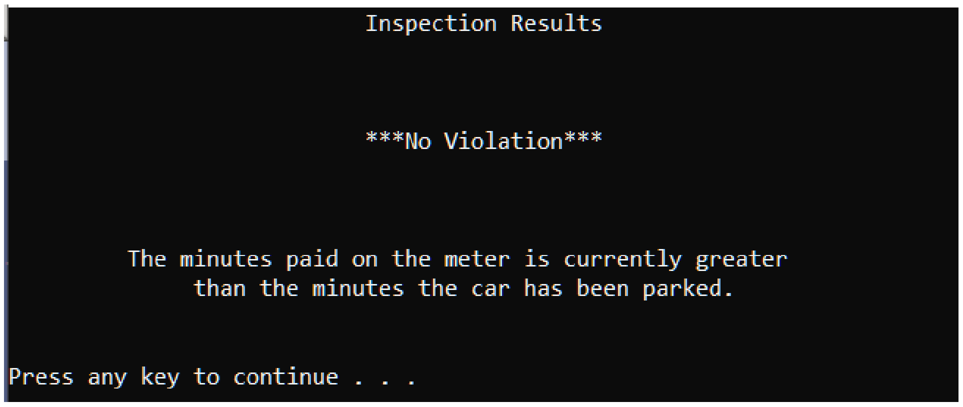
The PoliceOfficer Class contains a no args constructor and a constructor which takes parameters for all following variables . The setters and getters for each variable is listed.

Variable (private)	Type	Setter (public)	Getter (public)
officerName	string	`void setOfficerName(string);`	`string getOfficerName ();`
officerBadge	int	`void setOfficerBadge(int);`	`int getOfficerBadge ();`

Table 9 - PoliceOfficer Variables

examineCarMeter Function

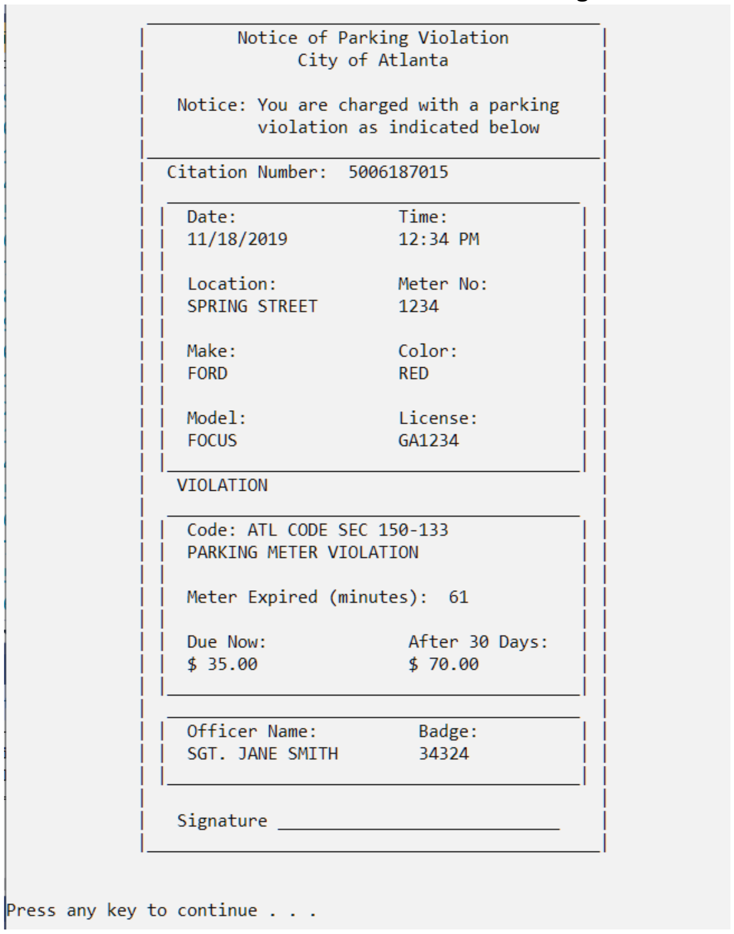
The function receives a ParkedCar object and a ParkingMeter object. The function compares the minParked variable of the ParkedCar to the minPaid of the ParkingMeter. If the minutes parked is less than the minutes paid. The function displays a message a ticket is not needed. If the minParked is greater than the minutes paid, the function calls the issueTicket function.



```
                    Inspection Results



                    ***No Violation***



     The minutes paid on the meter is currently greater
         than the minutes the car has been parked.


Press any key to continue . . .
```

issueTicket Function

The function receives a ParkedCar object and a ParkingMeter object from the calling function. The function creates a ParkingTicket object with the ParkedCar, ParkingMeter, and a copy of the PoliceOfficer. The function then calls the ParkingTicket function to print the ticket.



```
             _____________________________________________
            |         Notice of Parking Violation         |
            |               City of Atlanta               |
            |                                             |
            |   Notice: You are charged with a parking    |
            |           violation as indicated below      |
            |_____________________________________________|
            |  Citation Number:  5006187015               |
            |  _________________________________________  |
            | |  Date:                 Time:            | |
            | |  11/18/2019            12:34 PM         | |
            | |                                         | |
            | |  Location:             Meter No:        | |
            | |  SPRING STREET         1234             | |
            | |                                         | |
            | |  Make:                 Color:           | |
            | |  FORD                  RED              | |
            | |                                         | |
            | |  Model:                License:         | |
            | |  FOCUS                 GA1234           | |
            | |_________________________________________| |
            |   VIOLATION                                 |
            |  _________________________________________  |
            | |  Code: ATL CODE SEC 150-133             | |
            | |  PARKING METER VIOLATION                | |
            | |                                         | |
            | |  Meter Expired (minutes):  61           | |
            | |                                         | |
            | |  Due Now:               After 30 Days:  | |
            | |  $ 35.00                $ 70.00         | |
            | |_________________________________________| |
            |  _________________________________________  |
            | |  Officer Name:            Badge:        | |
            | |  SGT. JANE SMITH          34324         | |
            | |_________________________________________| |
            |                                             |
            |   Signature ____________________________    |
            |_____________________________________________|


Press any key to continue . . .
```

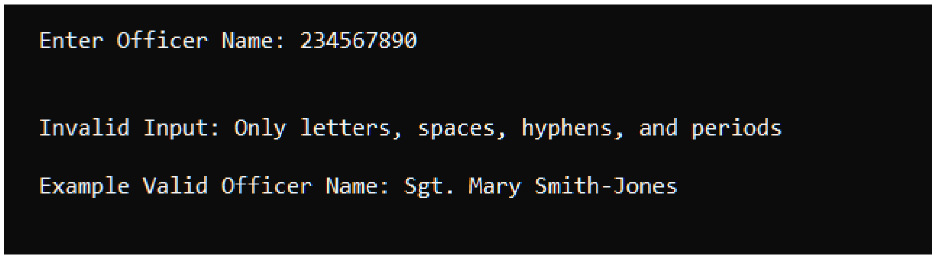
Friend Functions of the PoliceOfficer Class

operator << Overloaded Function

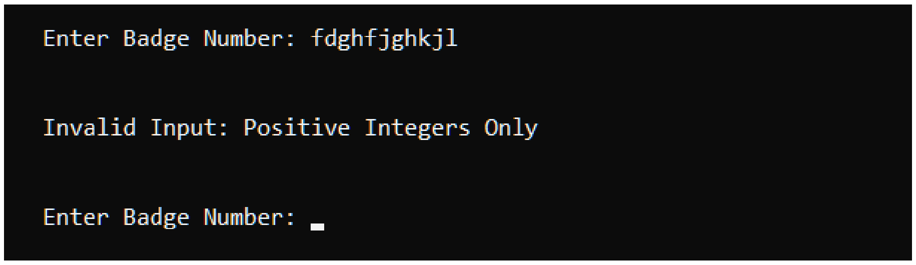
Takes an ostream reference object and constant PoliceOfficer reference object and returns a reference to the PoliceOfficer variables in the specified format. This allows cout to display the private member variables values without using the dot operator.

operator >> Overloaded Function

Takes an istream reference object and PoliceOfficer reference object and returns a reference to the police officer name and badge number. Input is validated using the Validation Class.



```
Enter Officer Name: 234567890


Invalid Input: Only letters, spaces, hyphens, and periods

Example Valid Officer Name: Sgt. Mary Smith-Jones
```



```
Enter Badge Number: fdghfjghkjl


Invalid Input: Positive Integers Only


Enter Badge Number: _
```

Class Declaration

The class specification file (PoliceOfficer.h) contains the declaration statements for the variables and functions that are members are of the class.

```
#ifndef POLICEOFFICER_H
#define POLICEOFFICER_H

#include <iostream>
#include <string>
#include "ParkedCar.h"
#include "ParkingMeter.h"

using namespace std;

class PoliceOfficer {

private:
	string officerName;			//string for officer name
	int officerBadge;			//integer for officer badge

public:
	//constructor - no args
	PoliceOfficer();

	//constructor
	PoliceOfficer(string, int);

	//copy constructor
	PoliceOfficer(const PoliceOfficer& officer2);

	//setters
	void setOfficerName(string);
	void setOfficerBadge(int);

	//getters
	string getOfficerName();
	int getOfficerBadge();

	//copy officer
	PoliceOfficer copyOfficer(const PoliceOfficer&);

	//examine car and meter
	void examineCarMeter(ParkedCar, ParkingMeter);

	//issue ticket
	void issueTicket(ParkedCar, ParkingMeter);

	//friends
	friend ostream& operator<<(ostream&, const PoliceOfficer&);
	friend istream& operator>>(istream&, PoliceOfficer&);
};

#endif
```
Table 10 – PoliceOfficer Declaration

Class Implementation

The class implementation file (PoliceOfficer.cpp) contains the definitions for the public functions of the class.

```
#include <iostream>
#include <string>
#include <iomanip>
#include <algorithm>

#include "PoliceOfficer.h"
#include "Validation.h"
#include "ParkingTicket.h"



//constructor - no args
PoliceOfficer::PoliceOfficer()
{
	officerName = "";
	officerBadge = 0;
}

//constructor
PoliceOfficer::PoliceOfficer(string n, int b)
{
	officerName = n;
	officerBadge = b;
}

// Copy constructor
PoliceOfficer::PoliceOfficer(const PoliceOfficer& officer2)
{
	officerName = officer2.officerName;
	officerBadge = officer2.officerBadge;
}

//set officer name
void PoliceOfficer::setOfficerName(string n)
{
	officerName = n;
}

//set officer badge
void PoliceOfficer::setOfficerBadge(int b)
{
	officerBadge = b;
}

//get officer name
string PoliceOfficer::getOfficerName()
{
	return officerName;
```

```
}

//get officer badge
int PoliceOfficer::getOfficerBadge()
{
	return officerBadge;
}

//copy officer
PoliceOfficer PoliceOfficer::copyOfficer(const PoliceOfficer& po)
{
	PoliceOfficer *copy = new PoliceOfficer(po);

	return *copy;
}

//examine car and meter
void PoliceOfficer::examineCarMeter(ParkedCar car, ParkingMeter meter)
{
	//clear console
	system("cls");

	//header
	cout << setw(47) << right << "Inspection Results\n\n";

	//if else to issue a ticket if meter expired
	if (car.getMinParked() > meter.getMinPaid())
	{
		cout << endl;
		cout << endl;

		cout << setw(52) << right << "***Parking Meter Violation***\n\n";

		cout << endl;

		//get the rest of the car information
		cin >> car;

		//issue a ticket
		issueTicket(car, meter);

	}

	else
	{
		cout << endl;
		cout << endl;

		cout << setw(47) << right << "***No Violation***\n\n";

		cout << endl;
		cout << endl;

		cout << setw(60) << right << "The minutes paid on the meter is currently
greater\n";
		cout << setw(57) << "than the minutes the car has been parked.\n\n"<<
endl;

		system("pause");
	}

}

//issue ticket
void PoliceOfficer::issueTicket(ParkedCar car, ParkingMeter meter)
{

	//create a parking ticket with car, meter, and officer
	ParkingTicket *ticket = new ParkingTicket(car, meter, copyOfficer(*this));

	//print the ticket
	ticket->printTicket();
}

//overload the cout <<
ostream& operator<<(ostream& strm, const PoliceOfficer& obj)
{
	//create strm of officer Name and Badge
	strm << "Officer Name: " << obj.officerName << endl
		<< "Badge Number: " << obj.officerBadge << endl;

	return strm;
}

//overload the cin >>
istream& operator>>(istream& strm, PoliceOfficer& obj)
{
	string name;				//input from user officer name
	string badgeStr;			//input from user badge as a string
	bool isValid = false;			//bool for valid officer info
	bool validName = false;			//bool for valid officer name
	bool validBadge = false;	//bool for valid officer badge
	bool validLength = false;	//bool for valid length

	//do to get officer information
	do
	{
		//try/catch for validation with exceptions
		try
		{
			do
			{
				//get valid name
				if (!validName)
				{
					cout << "\n   Enter Officer Name: ";
					getline(cin, name);

					if (name.empty())
					{
						throw Validation::EmptyInput();
					}

					validLength = Validation::isValidLength(name, 21);

					if (!validLength)
					{
						throw Validation::MaxLength();
```

```
                    }

                    transform(name.begin(), name.end(), name.begin(),
toupper);

                    validName = Validation::isValidName(name);

                    if (!validName)
                    {
                        throw Validation::InvalidInput();
                    }

                    validName = true;

                }

            }while (!validName);

            obj.setOfficerName(name);

            //get badge number
            do
            {
                cout << "\n   Enter Badge Number: ";
                getline(cin, badgeStr);

                if (badgeStr.empty())
                {
                    throw Validation::EmptyInput();
                }

                validBadge = Validation::isInteger(badgeStr);

                if (!validBadge)
                {
                    throw Validation::NumbersOnly();
                }

                validBadge = true;

            } while (!validBadge);

            obj.setOfficerBadge(stoi(badgeStr));

            isValid = true;
        }

        catch (Validation::InvalidInput)
        {
            cout << "\n\n   Invalid Input: Only letters, spaces, hyphens, and
periods" << endl;
            cout << "\n   Example Valid Officer Name: Sgt. Mary Smith-Jones\n"
<< endl;
        }

        catch (Validation::EmptyInput)
        {
            cout << "\n\n   Invalid Input: Field was empty\n" << endl;
        }
```

```
            catch (Validation::NumbersOnly)
            {
                  cout << "\n\n   Invalid Input: Positive Integers Only\n" << endl;
            }

            catch (Validation::MaxLength)
            {
                  cout << "\n\n   Invalid Input: Maximum Length of Input is 21
characters\n" << endl;
            }

            catch (out_of_range)
            {
                  cout << "\n\n   Max Length is 9 digits\n" << endl;
            }


      } while (!isValid);

      return strm;
}
```
Table 11 – Policer Officer Implementation

ParkingTicket Class:

The ParkingTicket Class contains a constructor which takes a ParkedCar, ParkingMeter and PoliceOfficer. The variables for the class and the setters and getters for each variable are listed.

Variable (private)	Type	Setter (public)	Getter (public)
carInfo	ParkedCar	`void setParkedCar(ParkedCar);`	`ParkedCar getParkedCar();`
meterInfo	ParkingMeter	`void setParkingMeter(ParkingMeter);`	`ParkingMeter getParkingMeter();`
officerInfo	PoliceOfficer	`void setPolicerOfficer(PoliceOfficer);`	`PoliceOfficer getPoliceOfficer();`
fine	double	`void setFine(int);`	`double getFine();`
BASE_FINE	const double	`BASE_FINE = 25.00;`	
ADD_FINE	const double	`ADD_FINE = 10.00;`	
citationNumber	string	`string getCitationNumber();`	`string getCitationNumber();`
location	string	`string getLocation();`	`string getLocation();`
meter	int	`int getMeterNumber();`	`int getMeterNumber();`

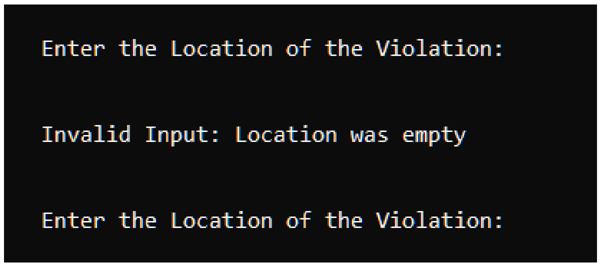
Table 12 - ParkingTicket Variables

getCitationNumber Function

The function uses a loop to randomly generate a 10-digit citation number.

getLocation Function

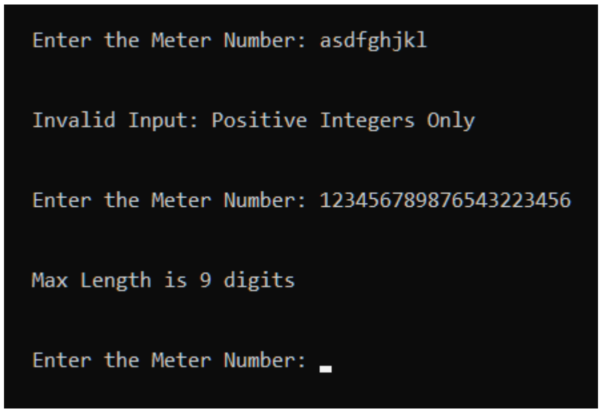
The function uses the Validation class, try/catch and a loop to get the location from the user. The validation only ensures the input is not empty.



```
Enter the Location of the Violation:

Invalid Input: Location was empty

Enter the Location of the Violation:
```

getMeter Function

The function uses the Validation class, try/catch and a loop to get the meter number from the user.



```
Enter the Meter Number: asdfghjkl

Invalid Input: Positive Integers Only

Enter the Meter Number: 123456789876543223456

Max Length is 9 digits

Enter the Meter Number: _
```

currentDayDateTime Function

The function gets the current date and time of the system to print on the ticket. The function uses the structure tm. The function prints the current date and time on the ticket.

calculateFine Function

The function calculates the amount of the fine using the minParked and minPaid. The amount is displayed on the ticket.

printTicket Function

The function calls the currentDayDateTime function and formats the ticket.

Class Declaration

The class specification file (ParkingTicket.h) contains the declaration statements for the variables and functions that are members are of the class.

```
#ifndef PARKINGTICKET_H
#define PARKINGTICKET_H

#include <ctime>
#include "ParkedCar.h"
#include "ParkingMeter.h"
#include "PoliceOfficer.h"

using namespace std;

class ParkingTicket {
private:
	ParkedCar carInfo;
	ParkingMeter meterInfo;
	PoliceOfficer officerInfo;
	double fine;
	const double BASE_FINE = 25.00;
	const double ADD_FINE = 10.00;
	string citationNumber;
	string location;
	int meter;


public:
	//constructor
	ParkingTicket(ParkedCar, ParkingMeter, PoliceOfficer);

	//setters
	void setParkedCar(ParkedCar);
	void setParkingMeter(ParkingMeter);
	void setPolicerOfficer(PoliceOfficer);
	void setFine(int);

	//getters
	ParkedCar getParkedCar();
	ParkingMeter getParkingMeter();
	PoliceOfficer getPoliceOfficer();
	double getFine();

	//get random citation number
	string getCitationNumber();

	//get location
	string getLocation();

	//get meter number
	int getMeterNumber();

	//get time
	static void currentDayDateTime();

	//calculate fine
	double calculateFine();
	//print ticket
	void printTicket();

};

#endif
```
Table 13 - ParkingTicket Declaration

Class Implementation

The class implementation file (ParkingTicket.cpp) contains the definitions for the public functions of the class.

```
#include "ParkingTicket.h"
#include <iomanip>
//Mickie Blair
//Final Project
//ParkingTicket Class
//Implementation File

#include <stdlib.h>
#include <algorithm>
#include <string>
#include "Validation.h"

using namespace std;

//constructor
ParkingTicket::ParkingTicket(ParkedCar c, ParkingMeter m, PoliceOfficer po)
{
	carInfo = c;
	meterInfo = m;
	officerInfo = po;
	fine = calculateFine();
	citationNumber = getCitationNumber();
	location = getLocation();
	meter = getMeterNumber();

}

//set parking ticket
void ParkingTicket::setParkedCar(ParkedCar c)
{
	carInfo = c;
}

//set parking meter
void ParkingTicket::setParkingMeter(ParkingMeter m)
{
	meterInfo = m;
}

//set police officer
void ParkingTicket::setPolicerOfficer(PoliceOfficer po)
{
	officerInfo = po;
}
```

```
//set fine
void ParkingTicket::setFine(int f)
{
	fine = f;
}

//get parked care
ParkedCar ParkingTicket::getParkedCar()
{
	return carInfo;
}

//get parking meter
ParkingMeter ParkingTicket::getParkingMeter()
{
	return meterInfo;
}

//get police officer
PoliceOfficer ParkingTicket::getPoliceOfficer()
{
	return officerInfo;
}

//get fine
double ParkingTicket::getFine()
{
	return fine;
}

//get random citation number
string ParkingTicket::getCitationNumber()
{
	string citNumber;
	int random;

	srand((unsigned int)time(NULL));

	for (int index = 0; index < 10; index++)
	{
		random = rand() % 9;
		citNumber.append(to_string(random));
	}

	return citNumber;
}

//get location
string ParkingTicket::getLocation()
{
	bool isValid = false;
	bool validLength = false;

	string street;

	do
	{
		try
		{
```

```
                        cout << "\n   Enter the Location of the Violation: ";
                        getline(cin, street);

                        if (street.empty())
                        {
                               throw Validation::EmptyInput();
                        }

                        validLength = Validation::isValidLength(street, 18);

                        if (!validLength)
                        {
                               throw Validation::MaxLength();
                        }

                        transform(street.begin(), street.end(), street.begin(), toupper);

                        isValid = true;
                 }

                 catch (Validation::EmptyInput)
                 {
                        cout << "\n\n   Invalid Input: Location was empty\n" << endl;
                 }

                 catch (Validation::MaxLength)
                 {
                        cout << "\n\n   Invalid Input: Maximum Length of Input is 18 
characters\n" << endl;
                 }


          } while (!isValid);

          return street;
}

//get meter number
int ParkingTicket::getMeterNumber()
{
          bool isValid = false;
          bool isInteger = true;

          string meterNo;
          int meterNumber;

          do
          {
                 try
                 {
                        cout << "\n   Enter the Meter Number: ";
                        getline(cin, meterNo);


                        if (meterNo.empty())
                        {
                               throw Validation::EmptyInput();
                        }
```

```
                    isInteger = Validation::isInteger(meterNo);

                    if (!isInteger)
                    {
                           throw Validation::NumbersOnly();
                    }

                    isInteger = true;

                    meterNumber = stoi(meterNo);

                    isValid = true;
             }

             catch (Validation::EmptyInput)
             {
                    cout << "\n\n   Invalid Input: Field was empty\n" << endl;
             }

             catch (Validation::NumbersOnly)
             {
                    cout << "\n\n   Invalid Input: Positive Integers Only\n" << endl;
             }

             catch (out_of_range)
             {
                    cout << "\n\n   Max Length is 9 digits\n" << endl;
             }


      } while (!isValid);

      return meterNumber;
}

//current day, date and time
void ParkingTicket::currentDayDateTime()
{
      string ticketDate = "   ";
      string ticketTime = "   ";
      string ext;

      time_t current_time;
      struct tm  local_time;

      time(&current_time);
      localtime_s(&local_time, &current_time);

      int Year = local_time.tm_year + 1900;
      int Month = local_time.tm_mon + 1;
      int Day = local_time.tm_mday;

      int Hour = local_time.tm_hour;
      int Min = local_time.tm_min;
      int Sec = local_time.tm_sec;

      //set the string of the date
      if (Month < 10)
      {
```

```
		ticketDate.append("0");
		ticketDate.append(to_string(Month));
		ticketDate.append("/");
	}
	else
	{
		ticketDate.append(to_string(Month));
		ticketDate.append("/");
	}

	if (Day < 10)
	{
		ticketDate.append("0");
		ticketDate.append(to_string(Day));
		ticketDate.append("/");
	}
	else
	{
		ticketDate.append(to_string(Day));
		ticketDate.append("/");
	}

	ticketDate.append(to_string(Year));

	//set the time
	if (Hour < 12)
	{
		ticketTime.append(to_string(Hour));
		ticketTime.append(":");
		ext = "AM";
	}
	else if (Hour == 12)
	{
		ticketTime.append("12");
		ticketTime.append(":");
		ext = "PM";
	}

	else
	{
		ticketTime.append(to_string(Hour-12));
		ticketTime.append(":");
		ext = "PM";
	}

	if (Min < 10)
	{
		ticketTime.append("0");
		ticketTime.append(to_string(Min));
		ticketTime.append(" ");
	}
	else
	{
		ticketTime.append(to_string(Min));
		ticketTime.append(" ");
	}

	ticketTime.append(ext);
```

```
        cout << setw(21) << left << ticketDate;
        cout << setw(20) << left << ticketTime;
}

//calculate fine
double ParkingTicket::calculateFine() {
        double minutesParked = carInfo.getMinParked();
        double minutesPaid = meterInfo.getMinPaid();
        double minutesOver;
        double hoursOver;

        minutesOver = minutesParked - minutesPaid;

        hoursOver = ceil(minutesOver / 60);

        {
                if (minutesOver <= 60)
                {
                        fine = BASE_FINE;
                }
                else
                {
                        fine = BASE_FINE;

                        hoursOver -= 1;

                        fine += hoursOver * ADD_FINE;
                }
        }

        return fine;

}

//print ticket
void ParkingTicket::printTicket()
{
        system("cls");
        system("Color F0");
        cout << setw(61) << right << "  _____________________________________________
\n";

        cout << setw(14) << right << "|";
        cout << setw(36) << right << "Notice of Parking Violation";
        cout << setw(11) << right << "|\n";

        cout << setw(14) << right << "|";
        cout << setw(30) << right << "City of Atlanta";
        cout << setw(17) << right << "|\n";

        cout << setw(14) << right << "|" << setw(47) << right << "|\n";

        cout << setw(14) << right << "|";
        cout << setw(41) << right << "Notice: You are charged with a parking";
        cout << setw(6) << right << "|\n";

        cout << setw(14) << right << "|";
        cout << setw(39) << right << "violation as indicated below";
        cout << setw(8) << right << "|\n";
        cout << setw(61) << right << "
|_____________________________________________|\n";

        cout << setw(14) << right << "|" << "  Citation Number:  "
               << setw(25) << left << citationNumber << "|\n";

        cout << setw(14) << right << "|"
               << "  _________________________________________  "<< "|\n";

        cout << setw(16) << right << "| |";
        cout << setw(21) << left << "  Date:" << setw(20) << left
               << "  Time:" << "| |\n";

        cout << setw(16) << right << "| |";
        ParkingTicket::currentDayDateTime();
        cout << "| |\n";

        cout << setw(16) << right << "| |" << setw(45) << right << "| |\n";

        cout << setw(16) << right << "| |";
        cout << setw(23) << left << "  Location:" << setw(18) << left
              << "Meter No:" << "| |\n";

        cout << setw(16) << right << "| |";
        cout << "  " << setw(21) << left << location
              << setw(17) << left << meter << " | |\n";

        cout << setw(16) << right << "| |" << setw(45) << right << "| |\n";

        cout << setw(16) << right << "| |";
        cout << setw(23) << left << "  Make:" << setw(18) << left
              << "Color:" << "| |\n";

        cout << setw(16) << right << "| |";
        cout << "  " << setw(21) << left << carInfo.getMake()
              << setw(18) << left << carInfo.getColor() << "| |\n";

        cout << setw(16) << right << "| |" << setw(45) << right << "| |\n";

        cout << setw(16) << right << "| |";
        cout << setw(23) << left << "  Model:" << setw(18) << left
              << "License:" << "| |\n";

        cout << setw(16) << right << "| |";
        cout << "  " << setw(21) << left << carInfo.getModel()
              << setw(18) << left << carInfo.getLicense() << "| |\n";


        cout << setw(14) << right << "|" << " |_____________________"
               << "____________________| " << "|\n";

        cout << setw(14) << right << "|" << "   VIOLATION" << setw(35)
              << right << "  |\n";

        cout << setw(14) << right << "|" << "  ________________________"
               << "_________________  " << "|\n";

        cout << setw(14) << right << "|" << " |  Code: ATL CODE SEC 150-133"
```

```
                << setw(17) << right << "| |\n";

        cout << setw(14) << right << "|" << " |  PARKING METER VIOLATION"
                << setw(20) << right << "| |\n";

        cout << setw(16) << right << "| |" << setw(45) << right << "| |\n";

        cout << setw(14) << right << "|" << " |  Meter Expired (minutes):  "
               << setw(12) << left << carInfo.getMinParked() - meterInfo.getMinPaid()
                << " | |\n";

        cout << setw(16) << right << "| |" << setw(45) << right << "| |\n";

        cout << setw(16) << right << "| |";
        cout << setw(24) << left << "  Due Now:" << setw(14) << left
               << "After 30 Days:  ";
        cout << " | |\n";

        cout << setw(16) << right << "| |";
        cout << "  $ " << setw(20) << left << setprecision(2) << fixed
                << fine << "$ " << setw(14) << left
                << fine * 2;
        cout << " | |\n";

        cout << setw(14) << right << "|" << " |_____________________"
               << "____________________| " << "|\n";

        cout << setw(14) << right << "|" << "  ________________________"
               << "_________________  " << "|\n";

        cout << setw(16) << right << "| |";
        cout << setw(23) << left << "  Officer Name:" << setw(18) << left
               << "  Badge:   ";
        cout << "| |\n";

        cout << setw(16) << right << "| |";
        cout << "  " << setw(23) << left << officerInfo.getOfficerName()
               << setw(16) << left << officerInfo.getOfficerBadge();
        cout << "| |\n";

        cout << setw(14) << right << "|" << " |_____________________"
               << "____________________| " << "|\n";

        cout << setw(14) << right << "|" << setw(47) << right << "|\n";

        cout << setw(61) << right << " |   Signature ____________________________
|\n";

        cout << setw(61) << right << "
|_____________________________________________|\n";

        cout << endl;
        cout << endl;

        system("pause");
}
```
Table 14 - ParkingTicket Implementation

Validation Class:

The Validation Class contain no variable. The static functions are used to valid input. The class contains the exceptions that are thrown if validation fails.

Exception Classes:

The program throws the follow exceptions during the program run.

- InvalidInput
- EmptyInput
- LettersOnly
- NumbersOnly
- NumbersOrLetters
- MaxLength

Static Methods:

The program consists of static methods to validate the various inputs needed.

isInteger Function

The function checks for integers only in the string received. Returns true or false.

isValidName Function

The function check for letters, periods, hyphens, and spaces in the string received. Returns true or false.

isLetters Function

The function checks for letters only in the string received. Returns true or false.

isLettersOrNumbers Function

The function checks for letters or numbers only in the string received. Returns true or false.

isValidLength Function

The function checks for valid length based on the integer received of the required size. Returns true or false.

Class Declaration

The class specification file (Validation.h) contains the declaration statements for the variables and functions that are members are of the class.

```
//Mickie Blair
//Final Project
//Validation Class
//Class Specification File

#ifndef VALIDATION_H
#define VALIDATION_H

#include <iostream>
#include <string>
```

```
using namespace std;

class Validation {

public:

	//validate input as integer
	static bool isInteger(string);

	//validate input as string (letter/space/hyphen/periods acceptable)
	static bool isValidName(string);

	//validate input as letters
	static bool isLetters(string);

	//validate input as letters or numbers only
	static bool isLettersOrNumbers(string);

	//validate for length
	static bool isValidLength(string, int);

//Exception Class
class InvalidInput {};
class EmptyInput {};
class LettersOnly {};
class NumbersOnly {};
class NumbersOrLetters {};
class MaxLength {};
};

#endif
```
Table 15 – Validation Declaration

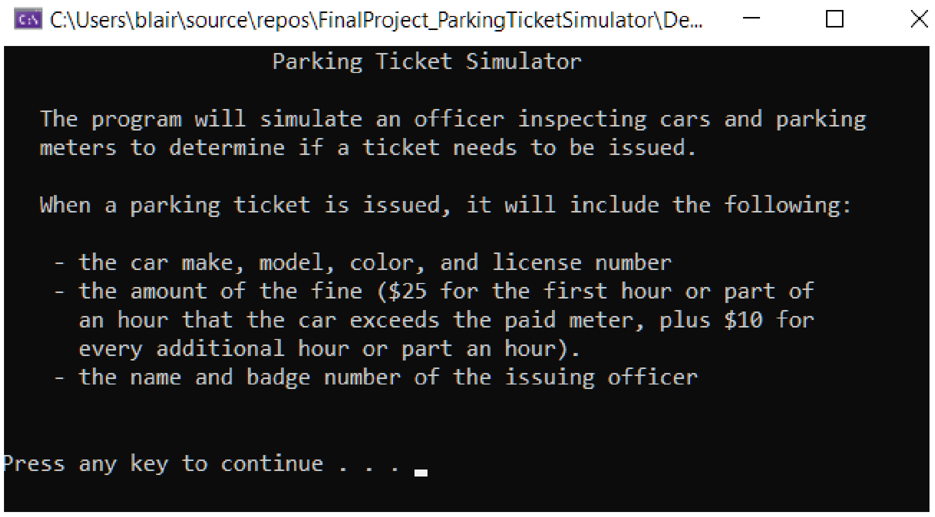
Class Implementation

The class implementation file (Validation.cpp) contains the definitions for the public functions of the class.

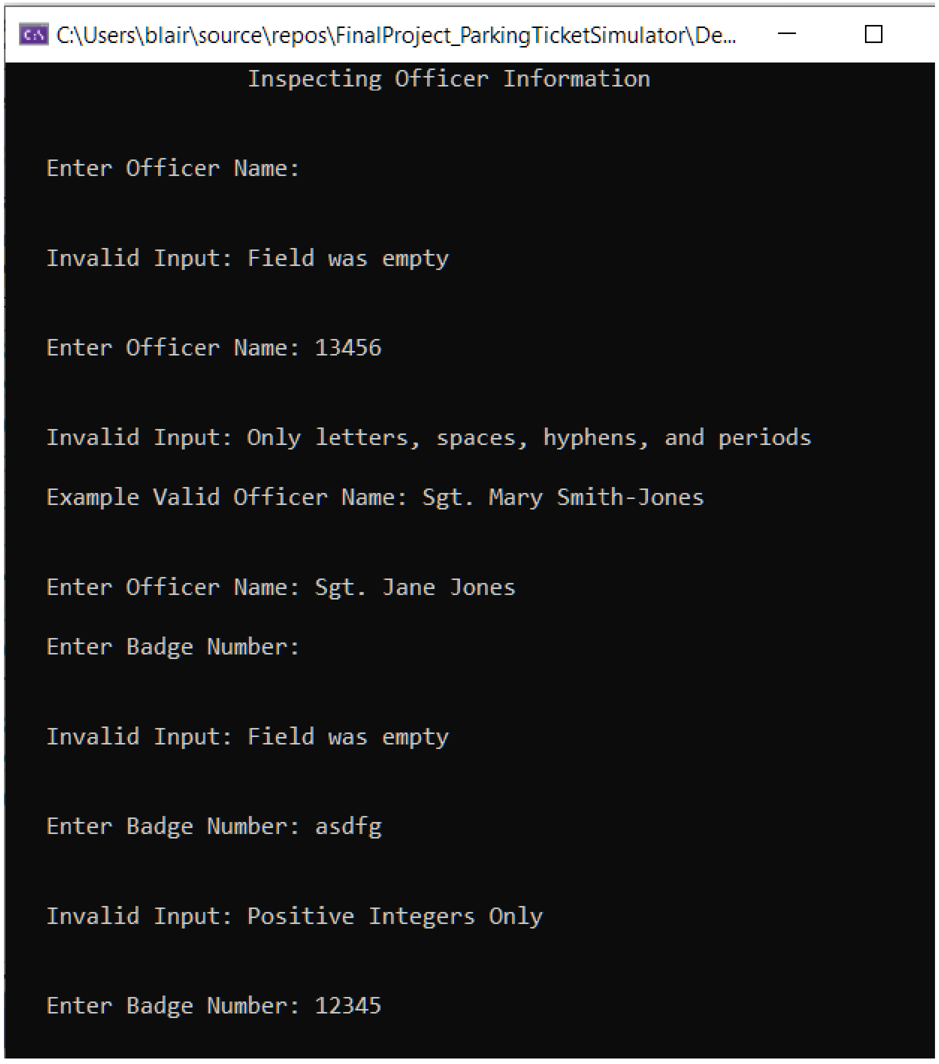
```
//Mickie Blair
//Final Project
//Validation Class
//Class Specification File

#include "Validation.h"

//validate for integer
bool Validation::isInteger(string numStr)
{
	bool isInteger = true;
	int index = 0;

	while (isInteger && index < int(numStr.length()))
	{
		if (!isdigit(numStr[index]))
			isInteger = false;
		index++;
	}
	return isInteger;
}

//validate for string length and letters/spaces/hyphens/periods only
bool Validation::isValidName(string str)
{
	bool isValid = false;
	bool hasLetters = false;
	bool validChars = true;
	int strIndex = 0;

	while (!hasLetters && strIndex < (int)str.length())
	{
		if (isalpha(str[strIndex]))
		{
			hasLetters = true;
		}

		strIndex++;
	}

	strIndex = 0;

	while (validChars && strIndex < (int)str.length())
	{
		if (!isalpha(str[strIndex]) && !isspace(str[strIndex])
			&& str[strIndex]!='-' && str[strIndex] != '.')
		{
			validChars = false;
		}

		strIndex++;
	}


	if (hasLetters && validChars)
	{
		isValid = true;
	}

	return isValid;
}

//validate for letters only
bool Validation::isLetters(string str)
{
	bool isLetters = true;
	int index = 0;

	while (isLetters && index < int(str.length()))
	{
		if(!isalpha(str[index]))
			isLetters = false;
		index++;
	}

	return isLetters;
}
```

```
//validate for string letters or numbers
bool Validation::isLettersOrNumbers(string str)
{
	bool validChars = true;
	int strIndex = 0;

	while (validChars && strIndex < (int)str.length())
	{
		if (!isalpha(str[strIndex]) && !isdigit(str[strIndex]))
		{
			validChars = false;
		}

		strIndex++;
	}

	return validChars;
}

//validate for max length
bool Validation::isValidLength(string str, int max)
{
	if (str.length() > max)
	{
		return false;
	}

	else
	{
		return true;
	}

}
```
Table 16 - Validation Implementation

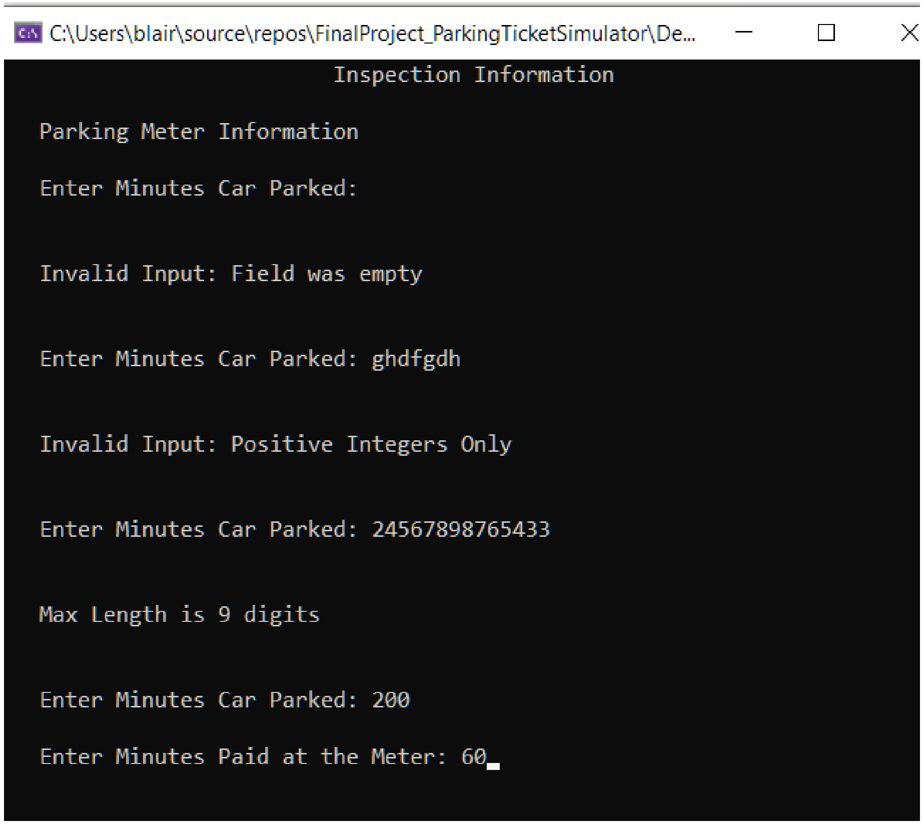
OUTPUT


C:\Users\blair\source\repos\FinalProject_ParkingTicketSimulator\De...

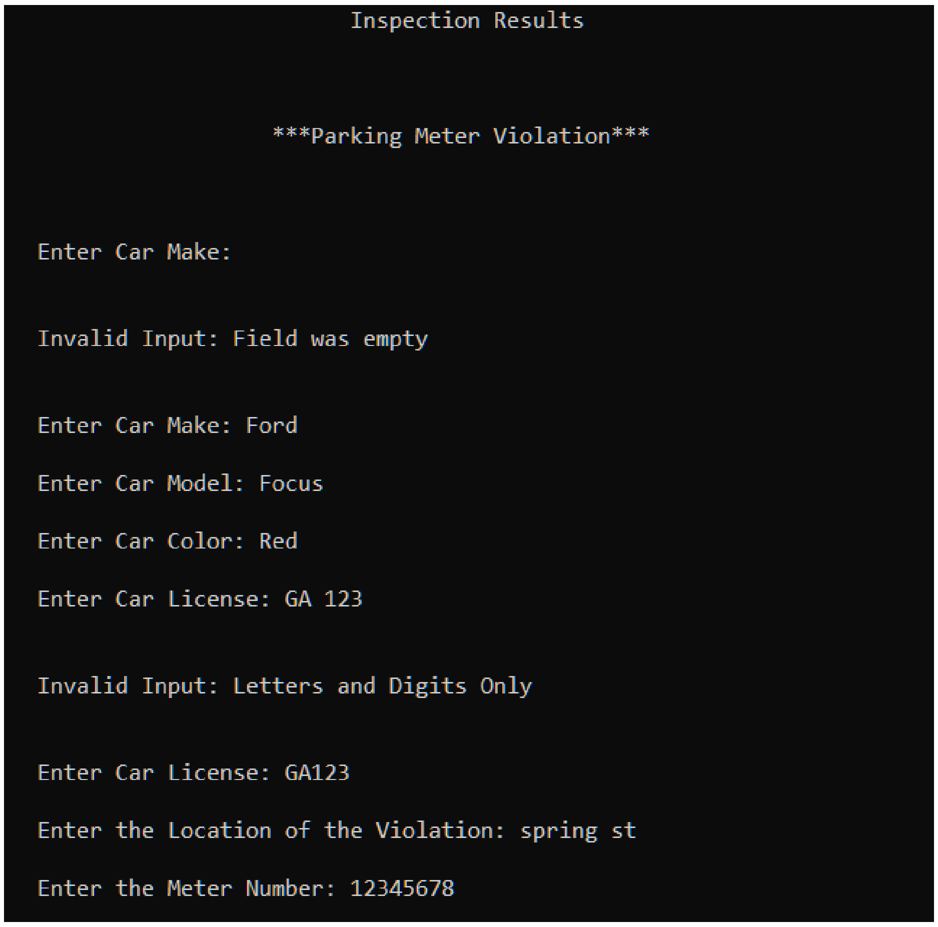
```
                    Parking Ticket Simulator

  The program will simulate an officer inspecting cars and parking
  meters to determine if a ticket needs to be issued.

  When a parking ticket is issued, it will include the following:

   - the car make, model, color, and license number
   - the amount of the fine ($25 for the first hour or part of
     an hour that the car exceeds the paid meter, plus $10 for
     every additional hour or part an hour).
   - the name and badge number of the issuing officer


Press any key to continue . . .
```


C:\Users\blair\source\repos\FinalProject_ParkingTicketSimulator\De...

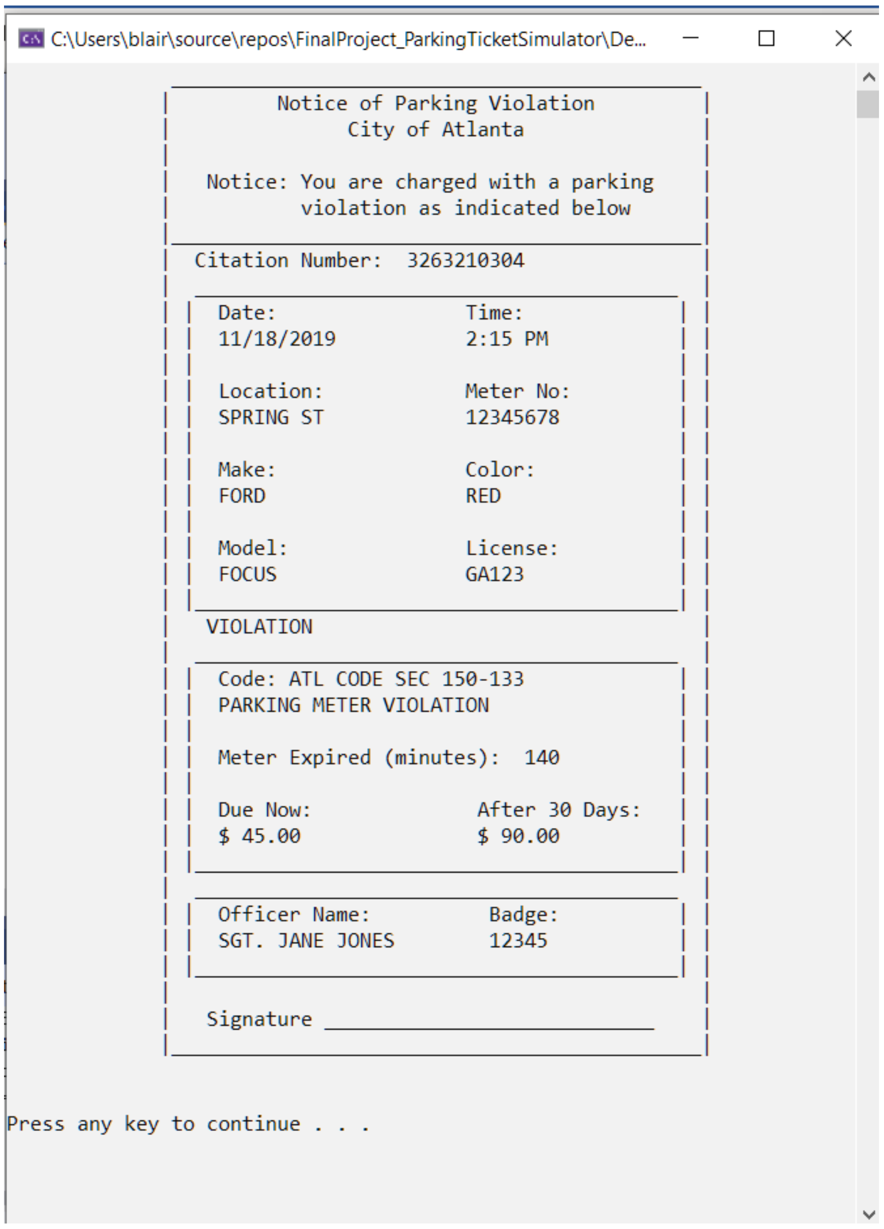
```
                  Inspecting Officer Information


   Enter Officer Name:


   Invalid Input: Field was empty


   Enter Officer Name: 13456


   Invalid Input: Only letters, spaces, hyphens, and periods

   Example Valid Officer Name: Sgt. Mary Smith-Jones


   Enter Officer Name: Sgt. Jane Jones

   Enter Badge Number:


   Invalid Input: Field was empty


   Enter Badge Number: asdfg


   Invalid Input: Positive Integers Only


   Enter Badge Number: 12345
```


C:\Users\blair\source\repos\FinalProject_ParkingTicketSimulator\De...

```
                            Inspection Information

 Parking Meter Information

 Enter Minutes Car Parked:


 Invalid Input: Field was empty


 Enter Minutes Car Parked: ghdfgdh


 Invalid Input: Positive Integers Only


 Enter Minutes Car Parked: 24567898765433


 Max Length is 9 digits


 Enter Minutes Car Parked: 200

 Enter Minutes Paid at the Meter: 60_
```

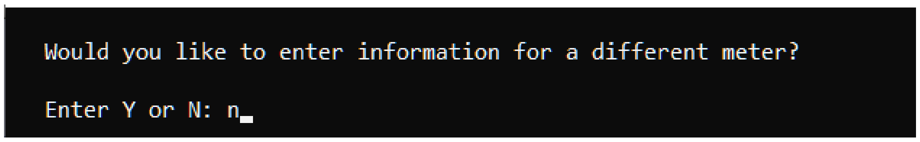


```
                            Inspection Results



                     ***Parking Meter Violation***



 Enter Car Make:


 Invalid Input: Field was empty


 Enter Car Make: Ford

 Enter Car Model: Focus

 Enter Car Color: Red

 Enter Car License: GA 123


 Invalid Input: Letters and Digits Only


 Enter Car License: GA123

 Enter the Location of the Violation: spring st

 Enter the Meter Number: 12345678
```

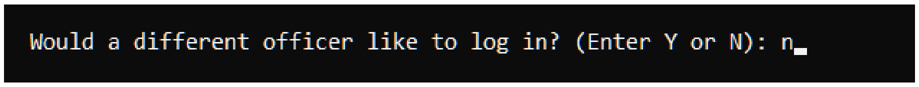


C:\Users\blair\source\repos\FinalProject_ParkingTicketSimulator\De...

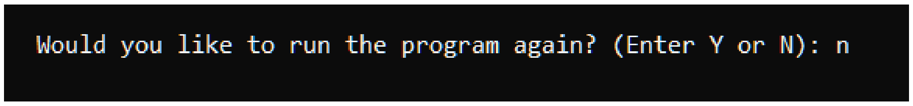
```
         ____________________________________________
        |        Notice of Parking Violation         |
        |              City of Atlanta               |
        |                                            |
        |   Notice: You are charged with a parking   |
        |           violation as indicated below     |
        |____________________________________________|
        |  Citation Number:  3263210304              |
        |   ________________________________________ |
        |  | Date:                Time:             ||
        |  | 11/18/2019           2:15 PM           ||
        |  |                                        ||
        |  | Location:            Meter No:         ||
        |  | SPRING ST            12345678          ||
        |  |                                        ||
        |  | Make:                Color:            ||
        |  | FORD                 RED               ||
        |  |                                        ||
        |  | Model:               License:          ||
        |  | FOCUS                GA123             ||
        |  |________________________________________||
        |   VIOLATION                                |
        |   ________________________________________ |
        |  | Code: ATL CODE SEC 150-133             ||
        |  | PARKING METER VIOLATION                ||
        |  |                                        ||
        |  | Meter Expired (minutes):  140          ||
        |  |                                        ||
        |  | Due Now:              After 30 Days:   ||
        |  | $ 45.00               $ 90.00          ||
        |  |________________________________________||
        |   ________________________________________ |
        |  | Officer Name:          Badge:          ||
        |  | SGT. JANE JONES        12345           ||
        |  |________________________________________||
        |                                            |
        |   Signature ____________________________   |
        |____________________________________________|


Press any key to continue . . .
```



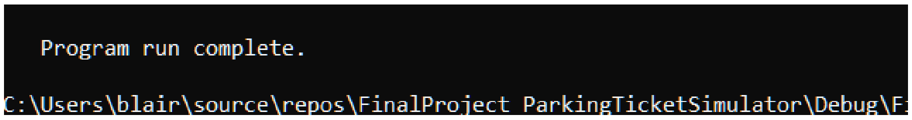
```
   Would you like to enter information for a different meter?

   Enter Y or N: n
```



```
  Would a different officer like to log in? (Enter Y or N): n
```



```
  Would you like to run the program again? (Enter Y or N): n
```



```
   Program run complete.

C:\Users\blair\source\repos\FinalProject_ParkingTicketSimulator\Debug\F
```

The Program Prototypes Header File

```
//Mickie Blair
//Final Project
//Prototypes Header File

//Prototypes Header file
#ifndef PROTOTYPES_H
#define PROTOTYPES_H

#include "PoliceOfficer.h"

using namespace std;

void showIntro();
void repeatProgram();
void parkingTicketDemo();
void inputOfficerInfo();
void getOfficerDetails(PoliceOfficer&);
void inspectCarMeter(PoliceOfficer&);

#endif
```

The Entire Program

```
//Mickie Blair
//Final Project
//Parking Ticket Simulator

#include "Prototypes.h"
#include "ParkedCar.h"
#include "ParkingMeter.h"
#include "ParkingTicket.h"
#include "PoliceOfficer.h"

#include <iostream>
#include <string>
#include <iomanip>
#include <new>
#include <algorithm>
#include <Windows.h>

using namespace std;

//main function
int main() {

	//set the console size
	HWND console = GetConsoleWindow();
	RECT r;

	GetWindowRect(console, &r);
	MoveWindow(console, r.top, r.left, 550, 750, TRUE);

	try
	{
		//repeat the program
		repeatProgram();

		//display complete message
		cout << "\n   Program run complete.\n";

	}
	catch (bad_alloc)
	{
		cout << "\n   Insufficient Memory.\n";
	}


	return 0;

}

//repeat Program
void repeatProgram() {

	string again;					//variable for again loop

	//do while loop
	do
	{
		//display introduction
		showIntro();

		//ParkingTicket Demo
		parkingTicketDemo();

		//ask user if they would like to go again
		cout << "\n   Would you like to run the program again? (Enter Y or N): ";
		getline(cin, again);

		//validate entry
		while (again != "Y" && again != "N" && again != "y" && again != "n")
		{
			cout << "\n   Invalid entry:" << endl;
		  cout << "\n   Would you like to run the program again? (Enter Y or N): ";
			getline(cin, again);
		}

		cout << endl;

		//clear
		system("CLS");
	}

	while (again == "Y" || again == "y");
}

//show Intro
void showIntro() {
	//display introduction
	cout << "                     Parking Ticket Simulator\n" << endl;
	cout << "   The program will simulate an officer inspecting cars and parking \n"
		<< "   meters to determine if a ticket needs to be issued.\n\n"
		<< "   When a parking ticket is issued, it will include the following:\n\n"
		<< "    - the car make, model, color, and license number \n"
		<< "    - the amount of the fine ($25 for the first hour or part of\n"
		<< "      an hour that the car exceeds the paid meter, plus $10 for \n"
		<< "      every additional hour or part an hour).\n"
```

```
              << "    - the name and badge number of the issuing officer\n\n"
              << endl;

       system("pause");
}

//parking ticket Demo
void parkingTicketDemo() {
       //clear the screen
       system("CLS");

       //get officer information with loop for new officer
       inputOfficerInfo();
}

//input Officer info with the option to choose a different issuing officer
void inputOfficerInfo() {

       string again;                              //variable for again loop
       PoliceOfficer officer;                     //officer object

       //do while loop
       do
       {
              cout << setw(50) << right << "Inspecting Officer Information\n\n";

              //get officer info
              getOfficerDetails(officer);

              //demonstrate the ticket simulation program
              inspectCarMeter(officer);

              system("cls");

              //change color
              system("Color 0F");

              //ask user if they would like to choose from the menu again
              cout << "\n   Would a different officer like to log in? (Enter Y or N): ";
              getline(cin, again);

              //validate entry
              while (again != "Y" && again != "N" && again != "y" && again != "n")
              {
                     cout << "\n\n   Invalid entry:" << endl;
              cout << "\n   Would a different officer like to log in? (Enter Y or N): ";
                     getline(cin, again);
              }

              //clear the screen
              system("CLS");
       }

       while (again == "Y" || again == "y");
}

//get officer info
void getOfficerDetails(PoliceOfficer & po) {
       //overloaded cin for officer
       cin >> po;

}

//demo ticket simulator to examine car and meter
void inspectCarMeter(PoliceOfficer& patrol) {
       system("cls");

       string again;                                   //variable for again loop
       ParkedCar car;                                          //parked car object
       ParkingMeter meter;                             //parking meter object

       //do while loop
       do
       {
              //header
              cout << setw(50) << right << "Inspection Information\n\n";

              //set the car minutes parked
              cout << "   Parking Meter Information \n";
              car.setMinParked();

              //set the meter
              cin >> meter;

              //officer examines car and meter
              patrol.examineCarMeter(car, meter);

              //clear the console
              system("cls");

              //change color
              system("Color 0F");

              //ask user if they would like to choose from the menu again
              cout << "\n   Would you like to enter information for"
                     << " a different meter?\n\n   Enter Y or N: ";
              getline(cin, again);

              //validate entry
              while (again != "Y" && again != "N" && again != "y" && again != "n")
              {
                     cout << "\n\n   Invalid entry:" << endl;
                     cout << "\n   Would you like to enter information for"
                            << " a different meter?\n\n   Enter Y or N: ";
                     getline(cin, again);
              }

              //clear the screen
              system("CLS");
       }

       while (again == "Y" || again == "y");

}
```