```
#include <iostream>
#include <iomanip>
#include <cstring>
#include "CarInventory.h"
using namespace std;
//-------------------------------------------------------------------------------
--------------------------------------------------------
namespace sdds
{
      CarInventory :: CarInventory()
      {
            CarInventory :: resetInfo();//member variables sets to nullptr and 0
respectively
      }
//-------------------------------------------------------------------------------
--------------------------------------------------------
      void CarInventory :: resetInfo()
      {
            m_type = nullptr, m_brand = nullptr, m_model = nullptr;
            m_year = 0, m_code = 0, m_price = 0;
      }
//-------------------------------------------------------------------------------
--------------------------------------------------------
      CarInventory :: CarInventory(const char* type, const char* brand, const char*
model, int year, int code, double price)
      {
            if (type != nullptr && model != nullptr && year >= 1990 && 100 <= code
&& code <= 999 && price > 0)
            {
                  //Allocation
                  m_type = new char[strlen(type) + 1], m_brand = new
char[strlen(brand) + 1], m_model = new char[strlen(model) + 1];

                  //strcpy(); copy one string to another
                  strcpy(m_type, type), strcpy(m_brand, brand), strcpy(m_model,
model);

                  //accessing private class members using arguments
                  m_year = year, m_code = code, m_price = price;
            }
            else
            {
                  resetInfo(); //m_type, m_brand, m_model sets to nullptr and
m_year, m_code, m_price sets to 0
            }
      }
//-------------------------------------------------------------------------------
--------------------------------------------------------
      CarInventory :: ~CarInventory()//Destruction
      {
            /*Dellocation of memory and clean-up for a class
            object and its class members*/
            delete[] m_type, delete[] m_brand, delete[] m_model;
            m_type = nullptr, m_brand = nullptr, m_model = nullptr;
      }
//-------------------------------------------------------------------------------
--------------------------------------------------------
      CarInventory& CarInventory :: setInfo(const char* type, const char* brand,
const char* model, int year, int code, double price)
      {
            //FIRST: Dellocation of memory
            delete[] m_type, delete[] m_brand, delete[] m_model;

            //SECOND: Request for memory allocation using keyword "new"
            m_type = new char[strlen(type) + 1], m_brand = new char[strlen(brand) +
1], m_model = new char[strlen(model) + 1];

            //strcpy(); copy one string to another
            strcpy(m_type, type), strcpy(m_brand, brand), strcpy(m_model, model);

            //accessing private class members using arguments
            m_year = year, m_code = code, m_price = price;
            return *this;
      }
//----------------------------------------------------------------------------------
-----------------------------------------------------
      void CarInventory :: printInfo() const
      {
    /*setw() is used to set the field width based on given width in the
parameter.*/
            cout << "| " << setw(10) << left << m_type;

            cout << " | " << setw(16) << left << m_brand;

            cout << " | " << setw(16) << left << m_model;

            cout << " | " << left << m_year;

            cout << " |  " << m_code;

            cout << " | " << fixed << setw(9) << setprecision(2) << right <<
m_price;

            cout << " |" << endl;

      }
//----------------------------------------------------------------------------------
-----------------------------------------------------
      bool CarInventory :: isValid() const
      {
            bool suppose{true};
            if (m_type != nullptr && m_brand != nullptr && m_model != nullptr &&
m_year >= 1990 && 100 <= m_code && m_code <= 999 && m_price > 0)
            {
                  suppose = true;
            }
            else
            {
                  suppose = false;
            }
            return suppose;
      }
//----------------------------------------------------------------------------------
-----------------------------------------------------
      bool CarInventory :: isSimilarTo(const CarInventory& car) const
      {
            bool suppose{true};
            if (car.isValid() && this->isValid())
            {
                  if (strcmp(m_type, car.m_type) == 0 && strcmp(m_brand,
car.m_brand) == 0 && strcmp(m_model, car.m_model) == 0)
                  {
                        suppose = false;
                  }
                  else
                  {
                        suppose = true;
                  }
            }
            else
            {
                  suppose = true;
            }
            return suppose;
      }
//------------------------------------------------------------------------------
------------------------------------------------------
      /*It returns true if it finds two CarInventory objects that have similar
information in the car array.*/
      bool find_similar(CarInventory car[], const int num_cars)
      {
            bool suppose{false};
            //function implementation logic:
            for (int i = 0; i < num_cars; i++) {
                  for (int j = i + 1; j < num_cars; j++) {
                        if (car[i].isSimilarTo(car[j]))
                        {
                              suppose = true;
                        }
                  }
            }
            return suppose;
      }
}
//------------------------------------------------------------------------------
------------------------------------------------------
```