



# PRACTICE 3

22/04/2025

---

Lydia Muñoz Gallardo

Aybaran Yurtseven  
Quality Software

## Index

<b>Index</b>	<b>1</b>
<b>Introduction</b>	<b>2</b>
<b>Standard 1</b>	<b>2</b>
<b>Standard 2</b>	<b>2</b>
<b>Standard 3</b>	<b>2</b>
<b>Standard 4</b>	<b>2</b>
<b>Standard 5</b>	<b>3</b>
<b>Standard 6</b>	<b>3</b>
<b>Standard 7</b>	<b>3</b>
<b>Standard 8</b>	<b>3</b>
<b>Standard 9</b>	<b>4</b>
<b>Standard 10</b>	<b>4</b>

## Introduction

The practice is about vehicles, it is a simple practice in C++ that has functions such as displaying vehicle data on the screen, or adding the weight of a certain number of vehicles. It was done in the Object Oriented Programming subject in 2022 by the student, Lydia.

## Standard 1

Link: DCL51-CPP. Do not declare or define a reserved identifier

Description:

The DCL51-CPP standard advises against declaring or defining reserved identifiers in C++ to avoid undefined behavior. Reserved identifiers include those with double underscores (`__`), those starting with an underscore followed by an uppercase letter, and various names from the standard library. It also prohibits redefining or undefining standard library names, keywords, and certain attribute tokens. These restrictions ensure compatibility, prevent conflicts, and maintain proper functionality within the language's standard implementation.

Code:

There is no "\_" at the beginning and at the end of the declaration of the "vehiculo.h" file.

```
5 // DCL51-CPP
6 #ifndef A_VEHICULO_H
7 #define A_VEHICULO_H
```

## Standard 2

Link: MEM51-CPP. Properly deallocate dynamically allocated resources

Description:

The MEM51-CPP standard emphasizes the importance of correctly releasing dynamically allocated resources in C++. When memory is allocated using `new`, it must be freed with `delete` once it is no longer needed. Failing to do so can lead to memory leaks, which may degrade the program's performance and stability. Similarly, any other dynamically allocated resource should be properly managed to prevent issues related to improper usage.

Code: There is the "delete" after using "new" with the pointer "v" in the code.

```

25 | //MEM51-CPP
26 | Vehiculo *v=new Vehiculo[tamv];|
27 |
28 | try{
29 |     rellenarVector (v,tamv);
30 | }catch(const string &e){
31 |     cerr<<e<<endl;
32 | }
33 |
34 | try{
35 |     cout<<" los vehiculos introducidos de 5 en 5 son :"<<endl;
36 |     mostrarEnPantalla( v, tamv );
37 | }catch(const string e){
38 |     cerr<<e;
39 | }
40 |
41 | mayor=MaxPrecio(v,tamv);
42 | cout<<" el vehiculo cuyo precio es mayor es : "<<endl;
43 | muestraEnPantalla(v[mayor]);
44 | try{
45 |     cout<<"introduce la matricula ";
46 |     cin>>matricula;
47 |     posicion= buscarPorMatricula( matricula, v,tamv);
48 |     cout<<" el vehiculo con matricula "<<matricula <<"ocupa la posicion"<<posicion<<endl;
49 |     LeePorTeclado(v[posicion]);
50 | }catch(const string &e){
51 |     cerr<<e<<endl;
52 | }
53 | try{
54 |     cargaEnTransporte(v,tamv);
55 | }catch(const string &e){
56 |     cerr<<e<<endl;
57 | }
58 |
59 | delete [] v;
60 | return 0;
61 | }

```

## Standard 3

Link: STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator

Description: Copying data into an undersized buffer causes a buffer overflow, often occurring with string manipulation. To prevent this, limit copies through truncation or ensure the destination has enough space. C-style strings require a null terminator, while `std::basic_string` in C++ does not.

Code: Fill the vector of elements and we do try catch so that if the buffer is full it does not introduce more elements

```
28 | //STR50-CPP
29 | try{
30 |     rellenarVector (v,tamv);
31 | }catch(const string &e){
32 |     cerr<<e<<endl;
33 | }
```

## Standard 4

Link: DCL52-CPP. Never qualify a reference type with `const` or `volatile`

Description: In C++, reference types cannot be modified, effectively treating all references as `const`. The C++ Standard states that cv-qualified references are invalid unless introduced via typedef or decltype, in which case the qualifiers are ignored. Only non-reference types can be cv-qualified. This restriction can lead to accidental misuse when trying to apply `const` to a reference type.

Code: We have got the reference of the object passed by parameter accompanying the object instead of the `const`

```
80 | //DCL52-CPP
81 | void vehiculos::mostrarEnPantalla(const Vehiculo *v, int tamv ){
```

## Standard 5

Link: [OOP53-CPP. Write constructor member initializers in the canonical order](#)

Description:

The OOP53-CPP standard emphasizes initializing class members in constructors following the order in which they are declared in the class definition. Maintaining this sequence improves code clarity, consistency, and readability, making it easier to maintain and understand. This practice is especially beneficial in classes with many members, as it ensures a predictable and structured approach to member initialization.

Code: As we can see we are initializing attributes in the default constructor in the same order we put them in the file.

```
26     string marca;  
27     string modelo;  
28     string matricula;  
29     int afabricacion;  
30     float precio;  
31     float peso;  
32 };  
  
49 //OOP53-CPP  
50 vehiculo::vehiculo(string modelo,string matricula, int afabricacion,float precio,float peso);
```

## Standard 6

Link: INT30-C. Ensure that unsigned integer operations do not wrap

Description:

The INT30-C standard ensures that unsigned integer operations do not result in overflow or wrapping. Unsigned integers in C are computed modulo  $2^N$ , which means that when the result of an operation exceeds the maximum value representable by the type, it wraps around. To prevent this, programmers should avoid situations where wrapping might occur, especially in pointer arithmetic, array indexing, or security-critical code. The standard also advises understanding integer conversion rules to ensure safe arithmetic operations.

Code:

Function partially complies with the INT30-C standard, but there is a potential risk if `p[i].precio` is a large value and `aux` is an unsigned integer (unsigned int), which could cause wrapping.

```
102 | //INT30-C (OLD)
103 | int vehiculos:: MaxPrecio(Vehiculo p[], int tamv){
104 |     int aux=0;
105 |     int pos=0;
106 |     for(int i=0; i<tamv; i++){
107 |         if(p[i].precio>aux){
108 |             aux=p[i].precio;
109 |             pos=i;
110 |         }
111 |     }
112 |     return pos;
113 | }
```

To ensure that additions or assignments do not cause wrapping, we add:

1. Use a larger data type (long long or uint64\_t if precio is unsigned int).
2. Ensure the value doesn't exceed the range of the data type.

```
102 //INT30-C (NEW)
103 ✓ int vehiculos::MaxPrecio(Vehiculo p[], int tamv) {
104     if (tamv <= 0) return -1; // Prevents out-of-bounds access
105     long long aux = 0; // Use a larger type to prevent wrapping
106     int pos = 0;
107
108     for (int i = 0; i < tamv; i++) {
109         ✓ if (static_cast<long long>(p[i].precio) > aux) { // Safe conversion
110             aux = static_cast<long long>(p[i].precio);
111             pos = i;
112         }
113     }
114     return pos;
115 }
```



## Standard 7

Link: [ERR56-CPP. Guarantee exception safety](#)

The text explains the importance of properly handling exceptions in C++ to ensure the stability and security of a program. It presents three levels of exception safety:

1. **Strong:** Guarantees that if an exception occurs, there are no changes to the program's state.
2. **Basic:** Prevents resource leaks and maintains program invariants.
3. **None:** Provides no exception safety guarantees, which can lead to an indeterminate state and critical errors.

It also includes an example of unsafe code and how to correct it to ensure strong exception safety. The corrected version first allocates memory for the copy before modifying the original object, preventing an exception from leaving the object in an inconsistent state.

```
33 //ERR56-CPP
34 Vehiculo& operator=( const Vehiculo &v){
35     int *a=nullptr;
36     if (v){
37         a= new Vehiculo;
38         a=v;
39     }
40     delete[] v;
41     v=a;
42     return *this;
43 }
```

## Standard 8

Link: [ERR52-CPP](#). Do not use `setjmp()` or `longjmp()`

The C standard library functions **`setjmp()`** and **`longjmp()`** can be used to simulate throwing and catching exceptions. However, these functions bypass automatic resource management and can lead to undefined behavior, commonly including memory leaks and denial-of-service attacks.

```
46 | //ERR52-CPP
47 | try{
48 |     cout<<"introduce la matricula ";
49 |     cin>>matricula;
50 |     posicion= buscarPorMatricula( matricula, v,tamv);
51 |     cout<<" el vehiculo con matricula "<<matricula <<"ocupa la posicion"<<posicion<<endl;
52 |     LeePorTeclado(v[posicion]);
53 | }catch(const string &e){
54 |     cerr<<e<<endl;
55 | }
56 | try{
57 |     cargaEnTransporte(v,tamv);
58 | }catch(const string &e){
59 |     cerr<<e<<endl;
60 | }
```

## Standard 9

Link: [DCL50-CPP. Do not define a C-style variadic function](#)

Variadic functions accept a variable number of arguments and can be defined in C++ using function parameter packs or C-style ellipses. However, C-style variadic functions are unsafe because they lack type checking and argument validation, leading to undefined behavior and potential security risks.

To avoid these issues, do not define C-style variadic functions. Instead, use function parameter packs or alternatives like function currying. For example, C++ replaces C's `printf()` with the type-safe overloaded `std::cout::operator<<()`.

Before:

```
void vehiculos:: mostrarEnPantalla(const Vehiculo *v, int tamv ){
    char c;
    for(int i=0; i<tamv; i++){
        muestraEnPantalla(v[i]);
        if((i+1)%5==0){
            cout<<"pulsa enter para los 5siguientes : ";
            cin>>c;
        }
    }
}
```

Modified:

```
83 void vehiculos:: mostrarEnPantalla(const Vehiculo *v, int tamv ){
84     char c;
85     for(int i=0; i<tamv; i++){
86         muestraEnPantalla(v[i]);
87         if((i+1)%5==0){
88             cout<<"pulsa enter para los 5siguientes : ";
89             cin.ignore(); //DCL50-CPP Instead of using `char c; cin >> c;`
90         }
91     }
92 }
93 }
```

## Standard 10

Link: [ERR62-CPP. Detect errors when converting a string to a number](#)

Parsing numbers from strings can lead to errors, such as invalid input, out-of-range values, or unexpected extra characters. These errors must be detected when using formatted input streams like `std::istream` or `num_get<>`.

Conversion errors can be checked via `basic_ios::good()`, `bad()`, and `fail()` or handled through exceptions. According to the C++ Standard, if conversion fails or the value is out of range, `ios_base::failbit` is set.

Always check the error state after conversion instead of assuming success. Avoid unsafe functions like `std::atoi()` and `std::scanf()` that lack validation.

Before:

```
16     int tamv; ///< Vehicle array size
17     string matricula; ///< Car license plate entered by the user
18     int posicion,mayor;///< Auxiliary variables for search and comparison
19     Vehiculo p; ///< Auxiliary vehicle
20     do{
21         cout<<"introduce el tamaño del vector";
22         cin>>tamv;
23     }while(tamv<=0);
24
```

In practice, we simply displayed integers according to this standard, ensuring proper error detection and handling during conversion that's why we changed it.

Modified:

```
16 int tamv; ///< Vehicle array size
17 string matricula; ///< Car license plate entered by the user
18 int posicion,mayor;///< Auxiliary variables for search and comparison
19 Vehiculo p; ///< Auxiliary vehicle
20 do{
21     cout<<"introduce el tamaño del vector";
22     cin>>tamv;
23     //ERR62-CPP
24     if(cin.fail()){
25         cin.clear;
26         cin.ignore(numeric_limits<streamsize>::max(), ' ');
27     }
28 }while(tamv<=0);
```

## Errors:

Problems 66 errors, 4 warnings, 6 others					
Description	Resource	Path	Location	Type	
Errors (66 items)					
Field 'afabricacion' could not be resolved	vehiculo.cpp	/pract3E13	line 30	Semantic Error	
Field 'afabricacion' could not be resolved	vehiculo.cpp	/pract3E13	line 63	Semantic Error	
Field 'marca' could not be resolved	vehiculo.cpp	/pract3E13	line 14	Semantic Error	
Field 'marca' could not be resolved	vehiculo.cpp	/pract3E13	line 15	Semantic Error	
Field 'marca' could not be resolved	vehiculo.cpp	/pract3E13	line 15	Semantic Error	
Field 'marca' could not be resolved	vehiculo.cpp	/pract3E13	line 60	Semantic Error	
Field 'matricula' could not be resolved	vehiculo.cpp	/pract3E13	line 25	Semantic Error	
Field 'matricula' could not be resolved	vehiculo.cpp	/pract3E13	line 26	Semantic Error	
Field 'matricula' could not be resolved	vehiculo.cpp	/pract3E13	line 62	Semantic Error	
Field 'matricula' could not be resolved	vehiculo.cpp	/pract3E13	line 113	Semantic Error	
Field 'modelo' could not be resolved	vehiculo.cpp	/pract3E13	line 20	Semantic Error	
Field 'modelo' could not be resolved	vehiculo.cpp	/pract3E13	line 21	Semantic Error	
Field 'modelo' could not be resolved	vehiculo.cpp	/pract3E13	line 21	Semantic Error	
Field 'modelo' could not be resolved	vehiculo.cpp	/pract3E13	line 61	Semantic Error	
Field 'peso' could not be resolved	vehiculo.cpp	/pract3E13	line 39	Semantic Error	
Field 'peso' could not be resolved	vehiculo.cpp	/pract3E13	line 128	Semantic Error	
Field 'precio' could not be resolved	vehiculo.cpp	/pract3E13	line 33	Semantic Error	
Field 'precio' could not be resolved	vehiculo.cpp	/pract3E13	line 34	Semantic Error	
Field 'precio' could not be resolved	vehiculo.cpp	/pract3E13	line 64	Semantic Error	
Field 'precio' could not be resolved	vehiculo.cpp	/pract3E13	line 96	Semantic Error	
Field 'precio' could not be resolved	vehiculo.cpp	/pract3E13	line 97	Semantic Error	
Function 'buscarPorMatricula' could not be resolved	main.cpp	/pract3E13	line 36	Semantic Error	
Function 'cargaEnTransporte' could not be resolved	main.cpp	/pract3E13	line 43	Semantic Error	
Function 'LeePorTeclado' could not be resolved	main.cpp	/pract3E13	line 38	Semantic Error	
Function 'MaxPrecio' could not be resolved	main.cpp	/pract3E13	line 30	Semantic Error	
Function 'mostrarEnPantalla' could not be resolved	main.cpp	/pract3E13	line 25	Semantic Error	
Function 'muestraEnPantalla' could not be resolved	main.cpp	/pract3E13	line 32	Semantic Error	
Function 'rellenarVector' could not be resolved	main.cpp	/pract3E13	line 18	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 12	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 48	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 59	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 73	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 92	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 111	Semantic Error	
Member declaration not found	vehiculo.cpp	/pract3E13	line 125	Semantic Error	

Description	Resource	Path	Location	Type
Member declaration not found	vehiculo.cpp	/pract3E13	line 111	Semantic Error
Member declaration not found	vehiculo.cpp	/pract3E13	line 125	Semantic Error
Method 'length' could not be resolved	vehiculo.cpp	/pract3E13	line 15	Semantic Error
Method 'length' could not be resolved	vehiculo.cpp	/pract3E13	line 15	Semantic Error
Method 'length' could not be resolved	vehiculo.cpp	/pract3E13	line 21	Semantic Error
Method 'length' could not be resolved	vehiculo.cpp	/pract3E13	line 21	Semantic Error
Method 'length' could not be resolved	vehiculo.cpp	/pract3E13	line 26	Semantic Error
Symbol 'cerr' could not be resolved	main.cpp	/pract3E13	line 20	Semantic Error
Symbol 'cerr' could not be resolved	main.cpp	/pract3E13	line 27	Semantic Error
Symbol 'cerr' could not be resolved	main.cpp	/pract3E13	line 40	Semantic Error
Symbol 'cerr' could not be resolved	main.cpp	/pract3E13	line 45	Semantic Error
Symbol 'cin' could not be resolved	main.cpp	/pract3E13	line 13	Semantic Error
Symbol 'cin' could not be resolved	main.cpp	/pract3E13	line 35	Semantic Error
Symbol 'cout' could not be resolved	main.cpp	/pract3E13	line 12	Semantic Error
Symbol 'cout' could not be resolved	main.cpp	/pract3E13	line 24	Semantic Error
Symbol 'cout' could not be resolved	main.cpp	/pract3E13	line 31	Semantic Error
Symbol 'cout' could not be resolved	main.cpp	/pract3E13	line 34	Semantic Error
Symbol 'cout' could not be resolved	main.cpp	/pract3E13	line 37	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 20	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 24	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 31	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 37	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 40	Semantic Error
Symbol 'endl' could not be resolved	main.cpp	/pract3E13	line 45	Semantic Error
Symbol 'vehiculos' could not be resolved	main.cpp	/pract3E13	line 3	Semantic Error
Type 'string' could not be resolved	main.cpp	/pract3E13	line 8	Semantic Error
Type 'string' could not be resolved	main.cpp	/pract3E13	line 19	Semantic Error
Type 'string' could not be resolved	main.cpp	/pract3E13	line 26	Semantic Error
Type 'string' could not be resolved	main.cpp	/pract3E13	line 39	Semantic Error
Type 'string' could not be resolved	main.cpp	/pract3E13	line 44	Semantic Error
Type 'Vehiculo' could not be resolved	main.cpp	/pract3E13	line 10	Semantic Error
Type 'Vehiculo' could not be resolved	main.cpp	/pract3E13	line 15	Semantic Error
Type 'Vehiculo' could not be resolved	main.cpp	/pract3E13	line 15	Semantic Error

## “Marca” Unresolved

```
void vehiculos::LeePorTeclado (Vehiculo &p ){
    cout << "introduce la marca";
    getline(cin >> ws, p.marca);
    if (p.marca.length() < 3 || p.marca.length() > 20){
        throw string("vehiculos:: LeePorTeclado: has introducido mal la marca");
    }
}
```

Cppcheck does not find the brand definition because it does not recognize that p is of type vehiculos::Vehiculo, or because vehiculos is not in the current namespace.

```
void vehiculos::LeePorTeclado (vehiculos::Vehiculo &p ){
    cout << "introduce la marca";
    ...
}
```

## Function 'MaxPrecio' could not be resolved

```
}

```

```
    mayor=MaxPrecio(v,tamv);
    cout<<" el vehiculo cuyo precio es mayor es : "<<endl;
    muestraEnPantalla(v[mayor]);
    try{
```

The MaxPrice function is defined within the vehicles namespace, so it is not directly recognized.

```
    mayor=vehiculos::MaxPrecio(v,tamv);
    cout<<" el vehiculo cuyo precio es mayo
```

After this errors there are not more:

Problems 0 errors, 5 warnings, 6 others					
Description	Resource	Path	Location	Type	
Warnings (5 items)					
(cppcheck style) Exception should be caught by ref	main.cpp	/pract3E13	line 26	cppcheck Probl...	
(cppcheck style) The scope of the variable 'matricu	main.cpp	/pract3E13	line 8	cppcheck Probl...	
(cppcheck style) The scope of the variable 'posicio	main.cpp	/pract3E13	line 9	cppcheck Probl...	
(cppcheck style) Unused variable: p	main.cpp	/pract3E13	line 10	cppcheck Probl...	
Catching by reference is recommended 'const strin	main.cpp	/pract3E13	line 26	Code Analysis ...	
Infos (6 items)					
(cppcheck information) Active checkers: 167/856 (u	pract3E13		Unknown	cppcheck Probl...	
(cppcheck information) Include file: <iostream> nc	main.cpp	/pract3E13	line 1	cppcheck Probl...	
(cppcheck information) Include file: <iostream> nc	vehiculo.h	/pract3E13	line 9	cppcheck Probl...	
(cppcheck information) Include file: <string> not fc	vehiculo.h	/pract3E13	line 10	cppcheck Probl...	
(cppcheck information) Too many #ifdef configura	CMakeCCompil...	/pract3E13/cmake-b...	Unknown	cppcheck Probl...	
(cppcheck information) Too many #ifdef configura	CMakeCXXCom...	/pract3E13/cmake-b...	Unknown	cppcheck Probl...	

## Warnings:

### Const string &e

```
    try{
        cout<<" los vehiculos introdu
        mostrarEnPantalla( v, tamv );
    }catch(const string e){
```

Before.



After:

```
try{
    cout<<" los vehiculos introducidos de 5 en 5 son :"<<endl;
    mostrarEnPantalla( v, tamv );
}catch(const string &e){
    cerr<<e;
}
```

The scope of the variable 'matricula' can be reduced.

```
string matricula;
int posicion,mayor;
Vehiculo p;
do{
    cout<<"introduce el tamaño del vector";
    cin>>tamv;
}while(tamv<=0);
Vehiculo *v=new Vehiculo[tamv];

try{
    rellenarVector (v,tamv);
}catch(const string &e){
    cerr<<e<<endl;
}

try{
    cout<<" los vehiculos introducidos de 5 en 5 son :"<<endl;
    mostrarEnPantalla( v, tamv );
}catch(const string &e){
    cerr<<e;
}

mayor=vehiculos::MaxPrecio(v,tamv);
cout<<" el vehiculo cuyo precio es mayor es : "<<endl;
muestraEnPantalla(v[mayor]);
try{
    cout<<"introduce la matricula ";
    cin>>matricula;
    posicion= buscarPorMatricula( matricula, v,tamv);
    cout<<" el vehiculo con matricula "<<matricula <<"ocupa la posicion"<<posicion<<endl;
```

Cppcheck detects that we are declaring the variable `matricula` in a wider scope than necessary. That is, we define it at the beginning of `main`, but only use it later. This takes up memory prematurely and reduces code clarity.

```
try{
    string matricula;
    cout<<"introduce la matricula ";
    cin>>matricula;
    posicion= buscarPorMatricula( matricula, v,tamv);
    cout<<" el vehiculo con matricula "<<matricula <<"ocupa la posicion"<<posicion<<endl;
    LeePorTeclado(v[posicion]);
}catch(const string &e){
```

Now we don't have any problem or warning.

Description	Resource	Path	Location	Type	
> i Infos (6 items)					

## Additional problems:

### Null Pointer Dereference

The pointer `v` is initialized with `new`, so it is not null. Always access `v[i]` within a valid range (e.g., `i < tamv`)

```
}while(tamv<=0);
Vehiculo *v=new Vehiculo[tamv];
```

### Use-After-Free

I only delete[] `v`; at the end of the program, when I'm not using it anymore. There's no access to `v` after it's been freed.

```

    int posicion, mayor;
    Vehiculo p;
    do{
        cout<<"introduce el tamaño del vector:"<<endl;
        cin>>tamv;
    }while(tamv<=0);
    Vehiculo *v=new Vehiculo[tamv];

    try{
        rellenarVector (v,tamv);
    }catch(const string &e){
        cerr<<e<<endl;
    }

    try{
        cout<<" los vehiculos i
        mostrarEnPantalla( v, tamv);
    }catch(const string e){
        cerr<<e;
    }

    mayor=vehiculos::MaxPrecio;
    cout<<" el vehiculo cuyo precio es mayor es:"<<endl;
    muestraEnPantalla(v[mayor]);
    try{
        cout<<"introduce la matricula:"<<endl;
        cin>>matricula;
        posicion= buscarPorMatricula(matricula);
        cout<<" el vehiculo con la matricula " <<matricula<<endl;
        LeePorTeclado(v[posicion]);
    }catch(const string &e){
        cerr<<e<<endl;
    }
    try{
        cargaEnTransporte(v,tamv);
    }catch(const string &e){
        cerr<<e<<endl;
    }

    delete [] v;
    return 0;

```

## Memory Leaks

There is no unreleased dynamic allocation.

```

    int posicion, mayor;
    Vehiculo p;
    do{
        cout<<"introduce el tamaño del vector:"<<endl;
        cin>>tamv;
    }while(tamv<=0);
    Vehiculo *v=new Vehiculo[tamv];

    try{
        rellenarVector (v,tamv);
    }catch(const string &e){
        cerr<<e<<endl;
    }

    try{
        cout<<" los vehiculos i
        mostrarEnPantalla( v, tamv);
    }catch(const string e){
        cerr<<e;
    }

    mayor=vehiculos::MaxPrecio;
    cout<<" el vehiculo cuyo precio es mayor es:"<<endl;
    muestraEnPantalla(v[mayor]);
    try{
        cout<<"introduce la matricula:"<<endl;
        cin>>matricula;
        posicion= buscarPorMatricula(matricula);
        cout<<" el vehiculo con la matricula " <<matricula<<endl;
        LeePorTeclado(v[posicion]);
    }catch(const string &e){
        cerr<<e<<endl;
    }
    try{
        cargaEnTransporte(v,tamv);
    }catch(const string &e){
        cerr<<e<<endl;
    }

    delete [] v;
    return 0;
}

```

## Dangerous use of functions like strcpy, etc.

We use getline, which is safe and scales to the string size. We don't use strcpy, gets, sprintf, scanf("%s"), etc.

```
cout << "introduce el modelo";
getline(cin >> ws, p.modelo);
```

## Double Free

We don't have double delete.

```
48
49 delete [] v;
50 return 0;
51 }
```

## No 'return' statement in non-void function causes undefined behaviour

All our non-void functions are returning things of the type declared. For instance:

```
2 int vehiculos:: MaxPrecio(Vehiculo p[], int tamv){
3     int aux=0;
4     int pos=0;
5     for(int i=0; i<tamv; i++){
6         if(p[i].precio>aux){
7             aux=p[i].precio;
8             pos=i;
9         }
10    }
11    return pos;
12 }
```