**Workshop 6: STL Containers**

In this workshop, you store polymorphic objects in an STL container.

You are going to create an application that simulates an autoshop that sells various types of vehicles. This specific application, will focus on **cars** and **racecars**.

**Learning Outcomes**

Upon successful completion of this workshop, you will have demonstrated the abilities to:

* manage polymorphic objects using the vector container of the STL
* move a dynamically allocated object into a container
* code a range-based iteration on the objects in a container
* report and handle an exception

**Submission Policy**

The *in-lab* section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). **In order to get credit for the *in-lab* portion, you must be present in the lab for the entire duration of the lab.**

The *at-home* portion of the workshop is due on the day that is four days after your scheduled *in-lab* workshop (@ 23:59:59), **even if that day is a holiday**.

All your work (all the files you create or modify) must contain your name, Seneca email, student number and the date of completion (use the following template):

// Name:

// Seneca Student ID:

// Seneca email:

// Date of completion:

//

// I confirm that the content of this file is created by me,

// with the exception of the parts provided to me by my professor.

You are responsible to back up your work regularly.

**Late Submission Penalties**

The workshop can be submitted up to **1 (one) day** late (the day that is 5 days after the lab period); submissions received on this day are considered **late** and are subject to penalties:

* only *in-lab* portion submitted late (after the end of the lab period): 0 for *in-lab* portion, max 7/10 for the entire workshop.
* only *at-home* portion submitted late (more than 4 days after the lab period): max 4 for the *at-home* portion, max 7/10 for the entire workshop.
* both *in-lab* **and** *at-home* portions submitted late: max 4/10 for the entire workshop.
* when the submission closes, if the workshop is not complete, the mark for the entire workshop will be 0/10. The workshop is considered complete if there are two separate submissions (*in-lab* submission and *at-home* submission) containing the *in-lab code*, *at-home code* and *reflection*.

The submission is considered closed at the end of the day that is 5 (five) days after the lab period.

***In-Lab***

The in-lab portion of this workshop consists of modules:

* w6 (supplied)
* Vehicle (supplied)
* Car
* Autoshop
* Utilities

Enclose all your source code within the sdds namespace and include the necessary guards in each header file.

**Car Module**

This module defines a class that holds information about a single car.

***Private Data***

Design and code a class named Car, that inherits the interface Vehicle (supplied), and that should be able to store the following information (for each attribute, you can chose any type you think it's appropriate--you must be able to justify the decisions you have made):

* **maker**
* **condition**: a car can be **new**, **used**, or **broken** in need of repairs.
* **top speed**

***Public Members***

* a custom constructor that receives an object of type std::istream& as parameter. This constructor should be able to read a single record from the stream, extract all the information about a single car and store it in the attributes. Each record has the following format:
* TAG,MAKER,CONDITION,TOP\_SPEED
  + TAG, is a single character representing the type of vehicle: c or C for car. Any other tag is considered invalid.
  + MAKER, the name of the company that makes the car
  + CONDITION, a single character: n for **new**, u for **used**, and b for **broken**. Any other character is considered invalid.
  + TOP\_SPEED, an integer containing the maximum speed the vehicle can achieve. If the content of this field is not a number, then the field is considered invalid.
  + any space at the beginning/end of each field should be removed.
* std::string condition() const: a query that returns new, used or broken, representing the condition of the car
* double topSpeed() const
* void display(std::ostream& out) const, a query that inserts in the first parameter the content of the car object in the format | MAKER | CONDITION | TOP\_SPEED |, where
  + MAKER, field of size 10
  + CONDITION, field of size 6
  + TOP\_SPEED, field of size 6, with two significant digits after the period
  + see alignment in the sample output.

**Add any other function that is required by your design!**

For the in-lab portion, assume there are no invalid records in the file.

**Autoshop module**

This module defines a class that holds information about an autoshop that sells various types of vehicles.

***Private Data***

Design and code a class named Autoshop that should be able to store the following information:

* std::vector<Vehicle\*> m\_vehicles: a vector that store all the vehicles available at this autoshop.

***Public Members***

* Autoshop& operator +=(Vehicle\* theVehicle): adds the vehicle received as parameter into the m\_vehicles vector.
* void display(std::ostream& out) const: iterates over the vehicles stored in m\_vehicles and prints them into the parameter (use Vehicle::display()) using the format:
* --------------------------------
* | Cars in the autoshop! |
* --------------------------------
* VEHICLE
* VEHICLE
* ...
* --------------------------------

**To iterate over the elements of the vector, use STL iterators!**

**Utilities module**

This module should contain a single function that creates instances on the Vehicle hierarchy:

Vehicle\* createInstance(std::istream& in);

This function should extract data from the parameter; if the first non-blank character is c or C, the this function should **dynamically** create an instance of type Car passing the stream to the constructor, and return it to the client.

If the first non-blank character is anything else (or there are no more characters to extract), this function should return nullptr.

Because the input file contains two types of delimiters (\n for records, and , for the fields in a record), you can use the class std::stringstream (utilization of this class is not mandatory, the extraction can be achieved without using it).

When implementing the createInstance function, consider the following STL class:

* [std::stringstream](https://en.cppreference.com/w/cpp/io/basic_stringstream/basic_stringstream)

**w6 Module (supplied)**

The tester module for the in-lab portion has been supplied. **Do not modify the existing code!**

When doing the workshop, you are encouraged to write your own tests, focusing on a single implemented feature at the time, until you get all functionality in place.

**Sample Output**

When the program is started with the command (the file cars.txt is provided):

w6.exe cars.txt

the output should look like the one from the sample\_output.txt file.

Command Line:

--------------------------

1: w6.exe

2: cars.txt

--------------------------

--------------------------------

| Cars in the autoshop! |

--------------------------------

| Toyota | new | 157.00 |

| Honda | broken | 145.00 |

| Chrysler | new | 173.00 |

--------------------------------

**Submission (30%)**

To test and demonstrate execution of your program use the same data as shown in the output example.

Upload your source code to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/9.1.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 305XXX\_w6\_lab

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

***At-Home***

The *at-home* part of this workshop upgrades your *in-lab* solution to include one more module and update some existing modules:

* Racecar
* Car (to be updated)
* Autoshop (to be updated)
* Utilities (to be updated)
* w6 (to be updated)

**Racecar Module**

Add a Racecar module to your project. A Racecar is a Car that for a short period of time can boost its top speed by a certain percent (this class should inherit Car).

***Private Data***

* double m\_booster: the percentage (stored as a number between 0 an 1) by which this Racecar can boost its top speed.

***Public Members***

* Racecar(std::istream& in): calls the constructor from Car to build the subobject, and then it extracts the last field from the stream containing the booster bonus. The input format for a racecar is TAG,MAKER,CONDITION,TOP\_SPEED,BOOSTER
* void display(std::ostream& out) const: calls Car::display() to print the information about the car, and adds \* at the end. The format should be | MAKER | CONDITION | TOP\_SPEED |\*
* double topSpeed() const: returns the top speed of the car, including any booster effects.

This class should not have access to the attributes of the parent class.

**Car Module**

Update the Car module to handle invalid records, and generate exceptions when an invalid record is detected. This requires you to modify the custom constructor to detect the following situations:

* the token representing the condition the car is empty (no characters or only blanks):
  + consider that the car is **new**
* the token representing the condition of the car is a different character than n, u, or b:
  + generate an exception to inform the client that this record is invalid
* the token representing the top speed is not a number:
  + generate an exception to inform the client that this record is invalid

No need to change anything else.

**Utilities module**

Update the createInstance to build an instance of type Racecar if the first non-blank character extracted from the stream is r of R.

If the first non-blank character is not c, C, r, or R this function should generate an exception, informing the client that this record contains information about an unknown vehicle.

If there is no more information to be extracted from the stream, this function should return nullptr.

**Autoshop Module**

Update this module to include two additional public functions.

***Public Members***

* a destructor. This function should iterate over the objects stored in the vector, and delete each one of them (note that the in-lab portion has a memory leak because the dynamically allocated vehicles were not deleted anywhere).
* void select(T test, std::list<const Vehicle\*>& vehicles): a **template** function that iterates over the vehicles stored in the m\_vehicles, and adds to the second parameter all vehicles for which the test is true. The first parameter (test) can be a lambda expression, a pointer to a function, or a functor that matches the prototype:
* bool func(const sdds::Vehicle\*);

**Since this is a template function, it must be implemented in the header!** The class is not a template.

**w6 Module (partially supplied)**

This module has some missing parts. The missing parts are marked with TODO, describing what code you should add and where. **Do not modify the existing code, only add what is missing!**

**Sample Output**

When the program is started with the command (the files are provided):

w6.exe dataClean.txt dataMessy.txt

the output should look like the one from the sample\_output.txt file.

Command Line:

--------------------------

1: w6.exe

2: dataClean.txt

3: dataMessy.txt

--------------------------

--------------------------------

| Cars in the autoshop! |

--------------------------------

| Toyota | new | 157.00 |

| Jaguar | used | 295.20 |\*

| Honda | broken | 145.00 |

| Porche | used | 365.40 |\*

| Chrysler | new | 173.00 |

--------------------------------

Invalid record!

Invalid record!

Unrecognized record type: [t]

--------------------------------

| Cars in the autoshop! |

--------------------------------

| Toyota | new | 157.00 |

| Jaguar | used | 295.20 |\*

| Honda | broken | 145.00 |

| Porche | used | 365.40 |\*

| Chrysler | new | 173.00 |

| Alfa Romeo | new | 176.00 |

| Ford | new | 162.00 |

| Red Bull | broken | 346.56 |\*

| Ferrari | new | 367.33 |\*

--------------------------------

--------------------------------

| Fast Vehicles |

--------------------------------

| Porche | used | 365.40 |\*

| Red Bull | broken | 346.56 |\*

| Ferrari | new | 367.33 |\*

--------------------------------

--------------------------------

| Vehicles in need of repair |

--------------------------------

| Honda | broken | 145.00 |

| Red Bull | broken | 346.56 |\*

--------------------------------

**Reflection**

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time.**

Create a **text** file named reflect.txt that contains your detailed description of the topics that you have learned in completing this particular workshop and mention any issues that caused you difficulty and how you solved them. Include in your explanation—**but do not limit it to**—the following points:

* Why do you need to deallocate the dynamically allocated memory in your vector?
* When building an object from the input file, different things can go wrong: the record represents an unknown type of vehicle, or record contains invalid data. How can the function loadData() detect what went wrong? Can you think at another solution than the one you have implemented?
* In this workshop you had to use std::list<sdds::Vehicle\*> and std::vector<sdds::Vehicle\*> to store a collection of vehicles as pointers. Could you have used std::list<sdds::Vehicle> and std::vector<sdds::Vehicle> instead? **Justify your answer!**

**Quiz Reflection**

Add a section to reflect.txt called **Quiz X Reflection**. Replace the **X** with the number of the last quiz that you received and list all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

**Submission (30% for code, 40% for reflection)**

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload the source code and the reflection file to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/9.1.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 305XXX\_w6\_home

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.