# Workshop 3: Templates

In this workshop, you design and code several class templates and test them on different instantiations.

# ## Learning Outcomes

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

- design and code a class template

- template a class variable

- specialize a templated class variable for a particular type

- instantiate a template class

- specialize a member function of a templated class to process a particular type

- derive a templated class from another templated class

# ## Submission Policy

The workshop is divided into two coding parts and one non-coding part:

- \*Part 1\*: worth 0% of the workshop's total mark, is optional and designed to assist you in completing the second part.

- \*Part 2\*: worth 100% of the workshop's total mark, is due **on \*\*Sunday at 23:59:59\*\*** of the week of your scheduled lab. Submissions of \*Part 2\* that do not contain the \*reflection\* are not considered valid submissions and are ignored.

- \*reflection\*: non-coding part, to be submitted together with \*Part 2\*. The reflection does not have

marks associated to it, but can incur a \*\*penalty of max 40% of the whole workshop's mark\*\* if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

Every file that you submit must contain (as a comment) at the top \*\*your name\*\*, \*\*your Seneca email\*\*, \*\*Seneca Student ID\*\* and the \*\*date\*\* when you completed the work.

If the file contains only your work, or work provided to you by your professor, add the following message as a comment at the top of the file:

> I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

If the file contains work that is not yours (you found it online or somebody provided it to you), \*\*write exactly which part of the assignment are given to you as help, who gave it to you, or which

source you received it from.\*\* By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

## Compiling and Testing Your Program

All your code should be compiled using this command on `matrix`:

```bash

**/usr/local/gcc/9.1.0/bin/g++ -Wall -std=c++17 -g -o ws file1.cpp file2.cpp ...**

```

- `-Wall`: compiler will report all warnings

- `-std=c++17`: the code will be compiled using the C++17 standard

- `-g`: the executable file will contain debugging symbols, allowing \*valgrind\* to create better reports

- `-o ws`: the compiled application will be named `ws`

After compiling and testing your code, run your program as following to check for possible memory leaks (assuming your executable name is `ws`):

```bash

valgrind ws

```

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use \*diff\* available on `matrix`.

If, when you try to compile/submit the workshop on matrix, you encounter an error regarding `/lib64/libstdc++.so.6: version 'CXXABI\_1.3.9 not found'`, add the following line at the end of your `.bashrc` file (this is a hidden text file located in the home folder--`~`--of your matrix account):

```bash

export LD\_LIBRARY\_PATH=/usr/local/gcc/9.1.0/lib64:$LD\_LIBRARY\_PATH

```

After you edit the file, logout and login again--this problem should go away.

# ## Part 1 (0%)

This workshop consists of five modules:

* `w3` (supplied)
* `Set`
* `Pair`
* `SetSummable`
* `PairSummable`

The solution to Part 1 invokes the first three modules.

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

All the modules \*\*you create\*\* in this workshop must have only a header file. \*🗎 Explain in the reflection why we do not split a module into `\*.h` and `\*.cpp` like you did in the previous workshops.\*

In all classes that you create, you are allowed to add any \*\*private\*\* members that your design requires (without changing the specs)!

## ### `w3` Module (supplied)

\*\*Do not modify this module!\*\* Look at the code and make sure you understand how to instantiate a templated class.

## ### `Set` Module

This module represents a family of collections of elements of any data type (for example, sets of `int`s, or sets of `Student`s, etc.).

Design and code a class template named `Set`. Your template manages a statically allocated array of any datatype. The template parameters in order of their specification are:

- `T`: the type of any element in the collection

- `N`: the maximum number of elements in the collection (a non-type parameter; an integer without sign)

Your design distinguishes between

- the capacity of the collection (`N`) and

- the current number of elements stored in the collection. Initially the collection has no elements.

\*\*This module should **not** use or know the type `Pair`!!\*\*

### \*\*\*Public Members\*\*\*

* `size\_t size() const`: returns the current number of elements in the collection
* `const T& operator[](size\_t idx) const`: returns the element at index `idx` of the statically allocated array (assume that the parameter is valid).
* `void operator+=(const T& item)`: if the collection has capacity for another element, adds a copy of `item` to the collection. Otherwise, does nothing.

## ### `Pair` Module

This module represents a family of \*key-value\* pairs of any types.

Design and code a class template named `Pair`. Your template manages a single \*key-value\* pair. The template parameters identify the type of the key and the type of the value that constitute a `Pair` object:

- `K`: the type of the key

- `V`: the type of the value

\*\*This module should not use or know the type `Set`!!\*\*

\*\*\*Public Members\*\*\*

- default constructor

- `Pair(const K& key, const V& value)`: copies the values referred to by the parameters into the instance variables

- `const K& key() const`: returns the \*\*key\*\* component of the pair

- `const V& value() const`: returns the \*\*value\*\* component of the pair

- `void display(std::ostream& os) const`: inserts into stream `os`

the key and the value of the pair in the following format

```

KEY : VALUE<endl>

```

\*\*\*Free Helper\*\*\*

- `std::ostream& operator<<(std::ostream& os, const Pair<K, V>& pair)`: calls the member

function `display()` on `pair` to insert a pair into stream `os`.

### Sample Output

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

```

w3.exe sales.txt

```

the output should look like:

```

Command Line:

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

1: w3.exe

2: sales.txt

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

Detail Ticket Sales

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

Student : 25

Adult : 13

Student : 12

Adult : 6

Student : 5

Adult : 15

Adult : 1

Adult : 2

Adult : 3

Student : 1

```

### Test Your Code

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

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 345\_w3\_p1

```

and follow the instructions.

\*\*\*This part represents a milestone in completing the workshop and is not marked!\*\*\*

# ## Part 2 (100%)

The second part of this workshop upgrades your Part 1 solution to include

- alignment of the key and value output in pretty columnar format

- accumulation of the values stored in a `Set` object, for a specified key

To implement each upgrade, you will derive templated classes from your original

templated classes (one derived class from `Set` and one derived class from `Pair`)

and specialize the class derived from `Pair` as described below.

## ### `Pair` Module

Modify the `display()` member function in the `Pair` module to enable inclusion polymorphism on the `Pair` hierarchy.

No other changes are necessary to this module.

## ### `PairSummable` Module

A new module called `PairSummable` represents a `Pair` that has key alignment and summation functionality.

Derive the `PairSummable` class template from `Pair<K, V>`. Yout template receives 2 template parameters:

- `K`: the type of the key

- `V`: the type of the value

\*\*This module should not use or know the type `Set` or `SetSummable`!!\*\*

\*\*\*Static Private Members for `PairSummable`\*\*\*

- an object of type `V` that holds the \*initial value\* for a summation. The initial value varies with the type of the value in the key-value pair and will be defined separately.

- a variable of type `size\_t` that holds the minimum field width for pretty columnar output of key-value pairs (initialize it with 0). This is the minimum number of characters needed to display any key in a set of keys.

This value must be updated every time a new pair is added to the collection.

### \*\*\*Public Members for `PairSummable`\*\*\*

- default constructor

- `PairSummable(const K& key, const V& value = initial)`: calls the base class constructor passing the two parameters to it, and updates the field width if necessary. \*🗎 Explain in the reflection what `= initial` in the prototype means.\*

- This function assumes that the type `K` supports a function named `size()` that returns the number of characters required to display `key`. Use this function to determine if the field width must be updated.

- overload the `operator+=` to receive a reference to an unmodifiable `PairSummable` object, to add the value of the parameter to the value of the current object \*\*if both objects share the same key but not the address\*\*, and to return a reference to the current object. Otherwise, this function leaves the current object unchanged and returns a reference to it.

- overload the `display(std::ostream& os)` query to set the alignment to left and the field width to that for all `K` types, then call the base class version of `display(std::ostream& os)`, and finally restore the alignment to right.

### \*\*\*Specializations\*\*\*

- for `K = std::string` and `V = std::string`, the initial value for summation should be an empty string (`""`).

- for `K = std::string` and `V = int`, the initial value for summation should be 0.

- for `K = std::string` and `V = std::string`, the function `operator+=()` should concatenate the values stored using `, ` as a separator (use operator `+` to concatenate strings), for the result shown in the sample output below.

## ### `Set` Module

The `Set` module doesn't need any change.

## ### `SetSummable` Module

A new module called `SetSummable` represents a collection that is summable and filterable.

Derive the `SetSummable` class template from your `Set<T, N>` template. Your new template has 2 template parameters:

- `T`: the type of any element in the collection

- `N`: the maximum number of elements in the collection (an integer without sign)

In your design, type `T`:

- has a constructor that accepts the address of an unmodifiable C-style string, which specifies the filter.

- supports the `operator+=` operation, which applies the filter to the summation operation on `T`.

\*\*This module should not use or know the type `Pair` or `PairSummable`!!\*\*

\*\*\*Public Member for `SetSummable`\*\*\*

- `T accumulate(const std::string& filter) const`: this query accumulates all the elements in the collection that satisfy `filter` into a local object of type `T`.

- initialize the accumulator by calling the one-argument constructor for a local `T` object that satisfies `filter`.

- iterate over the collection and add each item's contribution to the accumulator.

- return the accumulator to the client.

### Sample Output

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

```

w3.exe products.txt sales.txt

```

the output should look like:

```

Command Line:

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

1: w3.exe

2: products.txt

3: sales.txt

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

Individual Index Entries

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

Groceries : tomatoes

Electronics : computer

Tools : hammer

Groceries : lettuce

Groceries : potatoes

Electronics : Multimedia\_Player

Electronics : HDD

Groceries : meat

Tools : jigsaw

Collated Index Entries

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

Tools: hammer, jigsaw

Groceries: tomatoes, lettuce, potatoes, meat

Electrnics:

Electronics: computer, Multimedia\_Player, HDD

Detail Ticket Sales

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

Student : 25

Adult : 13

Student : 12

Adult : 6

Student : 5

Adult : 15

Adult : 1

Adult : 2

Adult : 3

Student : 1

Student : 2

Adult : 5

Adult : 6

Summary of Ticket Sales

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

Student Tickets = 92.70

Adult Tickets = 169.83

Senior Tickets = 0.00

```

### 🗎 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 and the result is suggested to be at least 150 words in length.\*\*

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:

- the reason for specializing the `operator+=()` member function.

- the reason for specializing the initial value for a summation.

- the reason for defining the class variable outside the class definition.

To avoid deductions, refer to code in your solution as examples to support your explanations.

### Submission

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 345\_w3\_p2

```

and follow the instructions.

\*\*:warning: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.