Associative containers

1. Content

2. Objective

At the end of these exercises you should be able to use:

- The std::pair struct
- The associative containers

Also, you should know how to define the **operator<** for ordered containers of custom types.

We advise you to **make your own summary of topics** that are new to you.

3. Exercises

Give your **projects** the same **name** as mentioned in the title of the exercise, e.g. **OrderedBasics**. Other names will be rejected.

Always adapt the **window title**: it should mention the name of the project, your name, first name and group.

Create a blank solution:

- **Name: W10**
- **Location: 1DAExx_10_name_firstname**

3.1. OrderedBasics

Add a new project with name **OrderedBasics** to the **W10** solution.

Overwrite **OrderedBasics.cpp** by the given one.

Copy and add the **Player** class files.

In this application you'll explore the **ordered associative containers**: set, multiset, map and multimap.

3.1.1. Overview of the members

Member functions	Member functions	Member functions	Member functions
multiset::multiset	set::set	map::map	multimap::multimap
multiset::~multiset	set::~set	map::~map	multimap::~multimap
multiset::operator=	set::operator=	map::operator=	multimap::operator=
multiset::get_allocator	set::get_allocator	map::get_allocator	multimap::get_allocator
Iterators	**Iterators**	**Element access**	**Iterators**
multiset::begin	set::begin	map::at	multimap::begin
multiset::cbegin (C++11)	set::cbegin (C++11)	map::operator[]	multimap::cbegin (C++11)
multiset::end	set::end	**Iterators**	multimap::end
multiset::cend (C++11)	set::cend (C++11)	map::begin	multimap::cend (C++11)
multiset::rbegin	set::rbegin	map::cbegin (C++11)	multimap::rbegin
multiset::crbegin (C++11)	set::crbegin (C++11)	map::end	multimap::crbegin (C++11)
multiset::rend	set::rend	map::cend (C++11)	multimap::rend
multiset::crend (C++11)	set::crend (C++11)	map::rbegin	multimap::crend (C++11)
Capacity	**Capacity**	map::crbegin (C++11)	**Capacity**
multiset::empty	set::empty	map::rend	multimap::erase
multiset::size	set::size	map::crend (C++11)	multimap::size
multiset::max_size	set::max_size	**Capacity**	multimap::max_size
Modifiers	**Modifiers**	map::empty	**Modifiers**
multiset::clear	set::clear	map::size	multimap::clear
multiset::insert	set::insert	map::max_size	multimap::insert
multiset::emplace (C++11)	set::emplace (C++11)	**Modifiers**	multimap::emplace (C++11)
multiset::emplace_hint (C++11)	set::emplace_hint (C++11)	map::clear	multimap::emplace_hint (C++11)
multiset::erase	set::erase	map::insert	multimap::erase
multiset::swap	set::swap	map::insert_or_assign (C++17)	multimap::swap
multiset::extract (C++17)	set::extract (C++17)	map::emplace (C++11)	multimap::extract (C++17)
multiset::merge (C++17)	set::merge (C++17)	map::emplace_hint (C++11)	multimap::merge (C++17)
Lookup	**Lookup**	map::try_emplace (C++17)	**Lookup**
multiset::count	set::count	map::erase	multimap::count
multiset::find	set::find	map::swap	multimap::find
multiset::equal_range	set::equal_range	map::extract (C++17)	multimap::equal_range
multiset::lower_bound	set::lower_bound	map::merge (C++17)	multimap::lower_bound
multiset::upper_bound	set::upper_bound	**Lookup**	multimap::upper_bound
		map::count	
		map::find	
		map::equal_range	
		map::lower_bound	
		map::upper_bound	

3.1.2. Explore std::pair

The **map** and **multimap** containers store information using the **std::pair** struct. It is a template that stores two objects as a single unit (The std::pair struct template).

So let's first make some basic exercises on this type. Complete the TODO task list in the function **TestPair**:

- Create pair objects.
- Get access to the 2 members.

```
--> TestPair
There are many ways to create a pair

There are 2 ways to get the values of a pair
[Andromeda, 73.62]
[World of Warcraft, 91.89]
```

3.1.3. Explore the set container

The **set** container (set) orders its elements using the **operator<** of its elements.

Complete the TODO task list in the TestSetOf… functions. Different sets are considered.

a. TestSetOfIntegers

This function tests a set of integer numbers. These are the test results.

Print the elements using a range based for loop

```
--> Set of integer elements
The elements of the set
10 20 30 40 50 60 70 80 90 100
```

The **insert** function doesn't insert when the value is already in the set. Insert the value of the first element again. Print all the elements using a range based for loop.

```
--> Set of integer elements
The elements of the set
10 20 30 40 50 60 70 80 90 100

Inserting an already existing member
Calling insert(10)
The elements of the set
10 20 30 40 50 60 70 80 90 100
```

Insert a value that is not in the set. The set is automatically sorted.

```
Inserting a new member
Calling insert(15)
The elements of the set
10 15 20 30 40 50 60 70 80 90 100
```

The **erase** function removes an element in the set, and it returns a result.

```
Erasing the first element, result is:
Calling erase(10) results in 1 removed
The elements of the set
15 20 30 40 50 60 70 80 90 100
```

Remove the same value again and show that result.

```
Erasing this element again, result is:
Calling once again erase(10) results in 0 removed
```

Loop over the elements and erase the ones that are divisible by 3.

You might notice that an exception is thrown. This happens because the moment an element is erased, we can no longer use this range based for loop. We must break it and redo the loop until no more elements are being erased. But do experience the crash and fix it afterwards.

```
Erasing all the elements that are divisible by 3
The elements of the set
20 40 50 70 80 100
```

Later (Prog3), you will see that there is a better way to solve this problem using iterators.

b. TestSetOfPlayers

In this test function, a set of user defined type – **Player** - is created. The **Player::operator<** sorts the players per name. Change the operator code to sort them to score.

```
--> Set of Player objects
De schoenmaker An (10)
Janssens Bart (20)
Janssens Xavier (30)
Devolder Warre (1000)
```

3.1.4. Explore the multiset container

The **multiset** container (multiset) offers the same member functions as the **set** container. However duplicate elements are allowed; hence the **insert** function always succeeds, and the returned type is different, look up the insert member function on the internet.

The **erase** function erases all occurrences.

Complete the TODO task list mentioned in **TestMultiset** function.

This is printed on the console.

```
--> Multiset of integer numbers
These are the elements of the vector used as source: 10 20 30 40
Elements of the multiset: 10 20 30 40

Inserting 3 times the value 50
Verify that the multiset contains 3 times the value 50
Elements of the multiset: 10 20 30 40 50 50 50

Erase the value 50 that occurs multiple times in the multiset
erase(50) => 3 elements erased
Verify that the multiset doesn't contain the value 50 anymore
Elements of the multiset: 10 20 30 40
```

3.1.5. Explore the map container

The map container (map) offers almost the same functionality as the set container, but it contains (key, value) pair-elements and is sorted using the first element of the pair (key). Duplicate keys are not allowed.

However the map has an extra functionality, like the vector it has the **operator[]** and function **at** to access an element, but you need to use its key instead of an index.

a. Map of game rankings

Complete the TODO task list mentioned in the **TestMapOfGameRankings** function in which a map is used for game ratings: **key** is the name of the game, **value** is the rating. Not all the map functions are considered, only the ones that differ from the set container. However, don't hesitate to have a look at them and test the ones that are not clear for you.

Below some screenshots of the console.

- After creation and adding elements to the map

```
--> How to create a map and adding elements to it
These are the elements of the map after adding elements in 3 different ways
Andromeda, 73.62
Final Fantasy XIV, 78.54
Grand Theft Auto, 94.39
League of Legends, 79.16
Wild Hunt, 79.16
World of Warcraft, 91.89
```

- After changing the content of the map

```
--> How to change the content of a map
Remove an element using erase(key), key is 'Wild Hunt'
Add an element with an already existing key but another value and check what happens
- Add 'Andromeda' with rating value 10.0, using the [] operator
- Adding 'League of Legends' with rating value 20.0, using insert
Game 'League of Legends' already exists
- Adding 'Grand Theft Auto' with rating value 30.0, using emplace
Game 'Grand Theft Auto' already exists

These are the elements of the map after these change operations
Andromeda, 10
Final Fantasy XIV, 78.54
Grand Theft Auto, 94.39
League of Legends, 79.16
World of Warcraft, 91.89
```

- When getting the value of an element

```
--> How to get the value of an element
Get the score of an existing game in 3 different ways and print the scores
Ratings of the game Andromeda 10, 10, 10

Get the score of a not existing game in 3 different ways
Ratings of the game notExistingGame 0

These are the elements of the map after getting some scores
Andromeda, 10
Final Fantasy XIV, 78.54
Grand Theft Auto, 94.39
League of Legends, 79.16
World of Warcraft, 91.89
notExistingGame, 0
```

- After changing the key of an element

```
--> How to change the key of an element
Rename one of the games
These are the elements of the map after changing 'Final Fantasy XIV' into 'Final Fantasy'
Andromeda, 10
Final Fantasy, 78.54
Grand Theft Auto, 94.39
League of Legends, 79.16
World of Warcraft, 91.89
notExistingGame, 0
```

b. Map of cities

Complete the TODO task list mentioned in the **TestMapOfCities** function in which a map of cities is tested: **key** is the name of the city; **value** is the City object.

```
--> Cities map, key is the name of a city, value is the City object

Insert the cities from a given vector of City objects into the map using a loop
These are the elements of the vector:
Aalst 84000
Kortrijk 76000
Gent 255000
Antwerpen 504000
Namen 111000
Hasselt 77000

These are the elements of the map after the insert operation
Aalst (84000)
Antwerpen (504000)
Gent (255000)
Hasselt (77000)
Kortrijk (76000)
Namen (111000)
```

3.1.6. Explore the multimap container

The **multimap** container (multimap) offers the same member functions as the **map** container except for the **operator[]** and **at** function. Those don't exist.

As the name suggests, this container allows elements with the same key, hence the **insert** function always succeeds and the returned type is different from the map container, look it up on the internet.

Complete the TODO task list mentioned in the **TestMultimap** function in which an English-Dutch dictionary is created using a multimap container.

This is the result on the console.

```
smart     heftig
right     juist
smart     bijdehand
strange   onbekend
date      dadel
right     rechts
date      afspraak

These are the elements of the multimap after the insert operation
English   Dutch
------------------
date      datum
date      dadel
date      afspraak
right     juist
right     rechts
smart     heftig
smart     bijdehand
strange   vreemd
strange   onbekend

Erase the elements with a specific key
These are the elements of the multimap after the erase operation
date      datum
date      dadel
date      afspraak
smart     heftig
smart     bijdehand
strange   vreemd
strange   onbekend
```

3.2. UnorderedBasics

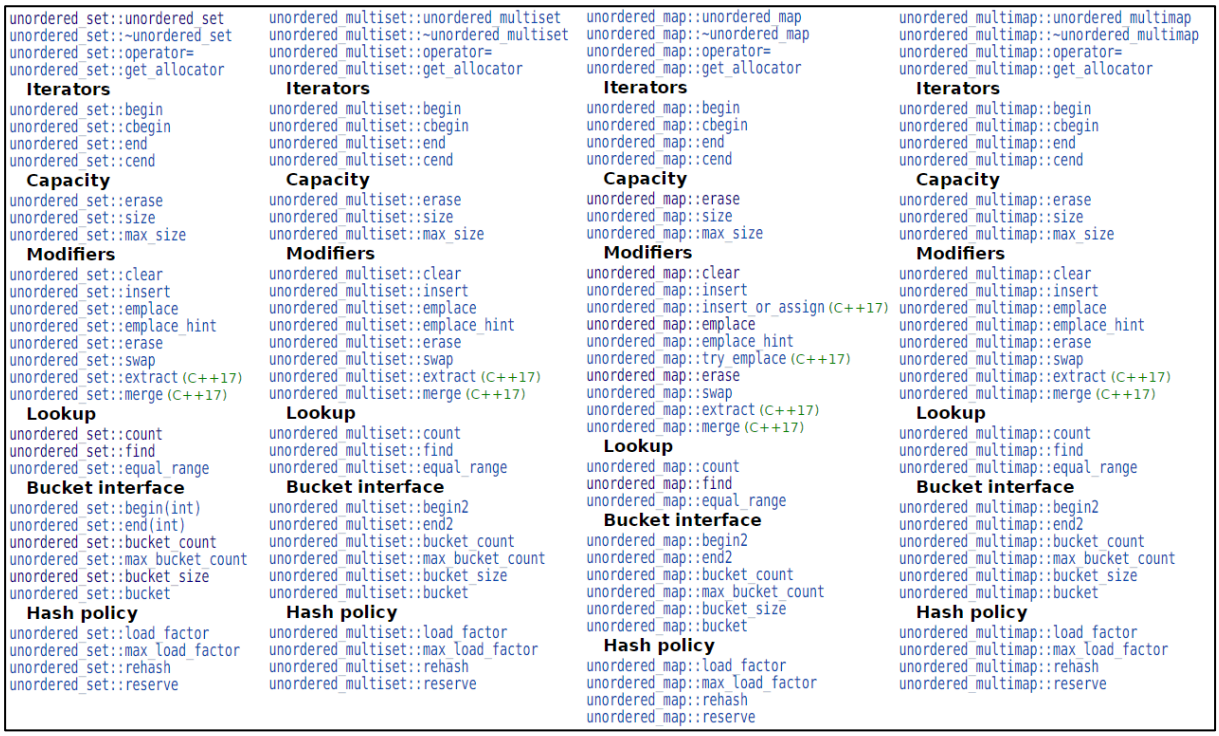
Add a new project with name **UnorderedBasics** to the **W10** solution.

Overwrite **UnorderedBasics.cpp** by the given one.

Copy and add the **Player** class files.

In this application you'll explore the **unordered associative containers**: unordered_set, unordered_multiset, unordered_map and unordered_multimap.

3.2.1. Overview of the members



unordered_set	unordered_multiset	unordered_map	unordered_multimap
unordered_set::unordered_set	unordered_multiset::unordered_multiset	unordered_map::unordered_map	unordered_multimap::unordered_multimap
unordered_set::~unordered_set	unordered_multiset::~unordered_multiset	unordered_map::~unordered_map	unordered_multimap::~unordered_multimap
unordered_set::operator=	unordered_multiset::operator=	unordered_map::operator=	unordered_multimap::operator=
unordered_set::get_allocator	unordered_multiset::get_allocator	unordered_map::get_allocator	unordered_multimap::get_allocator
Iterators	**Iterators**	**Iterators**	**Iterators**
unordered_set::begin	unordered_multiset::begin	unordered_map::begin	unordered_multimap::begin
unordered_set::cbegin	unordered_multiset::cbegin	unordered_map::cbegin	unordered_multimap::cbegin
unordered_set::end	unordered_multiset::end	unordered_map::end	unordered_multimap::end
unordered_set::cend	unordered_multiset::cend	unordered_map::cend	unordered_multimap::cend
Capacity	**Capacity**	**Capacity**	**Capacity**
unordered_set::erase	unordered_multiset::erase	unordered_map::erase	unordered_multimap::erase
unordered_set::size	unordered_multiset::size	unordered_map::size	unordered_multimap::size
unordered_set::max_size	unordered_multiset::max_size	unordered_map::max_size	unordered_multimap::max_size
Modifiers	**Modifiers**	**Modifiers**	**Modifiers**
unordered_set::clear	unordered_multiset::clear	unordered_map::clear	unordered_multimap::clear
unordered_set::insert	unordered_multiset::insert	unordered_map::insert	unordered_multimap::insert
unordered_set::emplace	unordered_multiset::emplace	unordered_map::insert_or_assign (C++17)	unordered_multimap::emplace
unordered_set::emplace_hint	unordered_multiset::emplace_hint	unordered_map::emplace	unordered_multimap::emplace_hint
unordered_set::erase	unordered_multiset::erase	unordered_map::emplace_hint	unordered_multimap::erase
unordered_set::swap	unordered_multiset::swap	unordered_map::try_emplace (C++17)	unordered_multimap::swap
unordered_set::extract (C++17)	unordered_multiset::extract (C++17)	unordered_map::erase	unordered_multimap::extract (C++17)
unordered_set::merge (C++17)	unordered_multiset::merge (C++17)	unordered_map::swap	unordered_multimap::merge (C++17)
Lookup	**Lookup**	unordered_map::extract (C++17)	**Lookup**
unordered_set::count	unordered_multiset::count	unordered_map::merge (C++17)	unordered_multimap::count
unordered_set::find	unordered_multiset::find	**Lookup**	unordered_multimap::find
unordered_set::equal_range	unordered_multiset::equal_range	unordered_map::count	unordered_multimap::equal_range
Bucket interface	**Bucket interface**	unordered_map::find	**Bucket interface**
unordered_set::begin(int)	unordered_multiset::begin2	unordered_map::equal_range	unordered_multimap::begin2
unordered_set::end(int)	unordered_multiset::end2	**Bucket interface**	unordered_multimap::end2
unordered_set::bucket_count	unordered_multiset::bucket_count	unordered_map::begin2	unordered_multimap::bucket_count
unordered_set::max_bucket_count	unordered_multiset::max_bucket_count	unordered_map::end2	unordered_multimap::max_bucket_count
unordered_set::bucket_size	unordered_multiset::bucket_size	unordered_map::bucket_count	unordered_multimap::bucket_size
unordered_set::bucket	unordered_multiset::bucket	unordered_map::max_bucket_count	unordered_multimap::bucket
Hash policy	**Hash policy**	unordered_map::bucket_size	**Hash policy**
unordered_set::load_factor	unordered_multiset::load_factor	unordered_map::bucket	unordered_multimap::load_factor
unordered_set::max_load_factor	unordered_multiset::max_load_factor	**Hash policy**	unordered_multimap::max_load_factor
unordered_set::rehash	unordered_multiset::rehash	unordered_map::load_factor	unordered_multimap::rehash
unordered_set::reserve	unordered_multiset::reserve	unordered_map::max_load_factor	unordered_multimap::reserve
		unordered_map::rehash	
		unordered_map::reserve	

3.2.2. Explore the unordered_set container

Complete the TODO task list mentioned in the **TestUnorderedSet** function in which an unordered_set of city names is created and tested (unordered_set).

```
--> Unordered_set of city names

Create and fill the unordered_set using a given the given vector of cities
These are the elements of the vector: Aalst Kortrijk Gent Antwerpen Aalst Namen Gent Hasselt
These are the elements of the unordered_set (with their hash value):
Aalst       681308222248448550
Hasselt     875106153383220343
Kortrijk   7630758819825406711
Antwerpen  4637678092449882915
Gent      10509118385549856019
Namen      7786727493982544280


Bucket information
Nr. buckets: 8, nr. elements: 6
Load factor: 0.75
Max load factor: 1
Bucket 0: Namen
Bucket 3: Antwerpen Gent
Bucket 6: Aalst
Bucket 7: Hasselt Kortrijk
```

And after adding the second range of cities.

```
Insert another vector of city names into the unordered_set
These are the elements of this vector: Brugge Mechelen Brussel Boom Lokeren
These are the elements of the unordered_set (with their hash value):
Aalst       681308222248448550
Kortrijk   7630758819825406711
Hasselt     875106153383220343
Antwerpen  4637678092449882915
Gent      10509118385549856019
Namen      7786727493982544280
Brugge     7155453871485214073
Boom      17851658315048716010
Mechelen   6380376538142920234
Brussel   11867228919478828039
Lokeren    9946228341375914469


Bucket information
Nr. buckets: 64, nr. elements: 11
Load factor: 0.171875
Max load factor: 1
Bucket 7: Brussel
Bucket 19: Gent
Bucket 24: Namen
Bucket 35: Antwerpen
Bucket 37: Lokeren
Bucket 38: Aalst
Bucket 42: Boom Mechelen
Bucket 55: Kortrijk Hasselt
Bucket 57: Brugge
```

After erasing the cities with name length >=8

```
Cities after erasing those with name length >= 8
Aalst
Hasselt
Gent
Namen
Brugge
Boom
Brussel
Lokeren
```

3.2.3. Explore the unordered_map container

Complete the TODO task list mentioned in the **TestUnorderedMap** function in which a unordered_map of Player objects is created and tested (unordered_map).

```
--> Unordered_map of Player objects, key is the name of the player

Create the map and then insert elements from the vector of Player objects
These are the elements of the vector:
Player[Jan, 3]
Player[Thomas, 12]
Player[Sara, 14]
Player[Kris, 8]
Player[Thomas, 6]
Player[Anna, 18]
Player[Sara, 19]

These are the  unordered_map elements together with their hash value
Player[Jan, 3] 6597105062029384174
Player[Thomas, 12] 7747982174193313405
Player[Kris, 8] 2653431476764356128
Player[Sara, 14] 3740885061712056496
Player[Anna, 18] 1470930550300165275


Bucket information
Nr. buckets: 8, nr. elements: 5
Load factor: 0.625
Max load factor: 1
Bucket 0: Player[Kris, 8]  Player[Sara, 14]
Bucket 3: Player[Anna, 18]
Bucket 5: Player[Thomas, 12]
Bucket 6: Player[Jan, 3]
```

After erasing the players with score < 10.

```
Players after erasing those with score < 10
Player[Thomas, 12]
Player[Sara, 14]
Player[Anna, 18]

Bucket information
Nr. buckets: 8, nr. elements: 3
Load factor: 0.375
Max load factor: 1
Bucket 0: Player[Sara, 14]
Bucket 3: Player[Anna, 18]
Bucket 5: Player[Thomas, 12]
```

3.2.4. Explore the unordered_multiset container

Complete the TODO task list mentioned in the **TestUnorderedMultiset** function in which a unordered_multiset of city names is created and tested (unordered_multiset).

```
--> unordered_multiset of city names

Create and fill the unordered_multiset using a given vector of cities
These are the elements of the vector: Aalst Kortrijk Gent Antwerpen Aalst Namen Gent Hasselt
These are the elements of the unordered_multiset (with their hash value):
Aalst       681308222248448550
Aalst       681308222248448550
Hasselt     875106153383220343
Kortrijk   7630758819825406711
Antwerpen  4637678092449882915
Gent      10509118385549856019
Gent      10509118385549856019
Namen      7786727493982544280


Bucket information
Nr. buckets: 8, nr. elements: 8
Load factor: 1
Max load factor: 1
Bucket 0: Namen
Bucket 3: Antwerpen Gent Gent
Bucket 6: Aalst Aalst
Bucket 7: Hasselt Kortrijk
```

After inserting another vector of cities.

```
Insert another vector of city names into the unordered_multiset
These are the elements of this vector: Brugge Mechelen Brussel Gent Hasselt
These are the elements of the unordered_multiset (with their hash value):
Aalst       681308222248448550
Aalst       681308222248448550
Kortrijk   7630758819825406711
Hasselt     875106153383220343
Hasselt     875106153383220343
Antwerpen  4637678092449882915
Gent      10509118385549856019
Gent      10509118385549856019
Gent      10509118385549856019
Namen      7786727493982544280
Brugge     7155453871485214073
Mechelen   6380376538142920234
Brussel   11867228919478828039


Bucket information
Nr. buckets: 64, nr. elements: 13
Load factor: 0.203125
Max load factor: 1
Bucket 7: Brussel
Bucket 19: Gent Gent Gent
Bucket 24: Namen
Bucket 35: Antwerpen
Bucket 38: Aalst Aalst
Bucket 42: Mechelen
Bucket 55: Kortrijk Hasselt Hasselt
Bucket 57: Brugge
```

3.2.5. Explore the unordered_multimap container

Complete the TODO task list mentioned in the **TestUnorderedMultimap** function in which an unordered_multimap of Player objects is created and tested (unordered_multimap).

```
--> Unordered_multimap of Player objects, key is the name of the player

Create the map and then insert elements from the vector of Player objects
These are the elements of the vector:
Player[Jan, 22515]
Player[Thomas, 11189]
Player[Sara, 9227]
Player[Kris, 32553]
Player[Thomas, 1185]
Player[Anna, 17970]
Player[Sara, 2120]

These are the unordered_multimap elements together with their hash value
Player[Jan, 22515] 6597105062029384174
Player[Thomas, 11189] 7747982174193313405
Player[Thomas, 1185] 7747982174193313405
Player[Kris, 32553] 2653431476764356128
Player[Sara, 9227] 3740885061712056496
Player[Sara, 2120] 3740885061712056496
Player[Anna, 17970] 1470930550300165275


Bucket information
Nr. buckets: 8, nr. elements: 7
Load factor: 0.875
Max load factor: 1
Bucket 0: Player[Kris, 32553]  Player[Sara, 9227]  Player[Sara, 2120]
Bucket 3: Player[Anna, 17970]
Bucket 5: Player[Thomas, 11189]  Player[Thomas, 1185]
Bucket 6: Player[Jan, 22515]
```

3.3. TextureManager

Create a new framework project in the solution with the name Texture manager.

Add a class with the name: TextureManager.

The task of this class is to own texture objects, pass their pointer to anyone who asks for it. Any object in a project that has a reference to an object of this class, can use this class to get Texture pointers. This prevents that a texture is loaded more than one time, saving gpu memory and centralizing the Texture managing.

Add a member function `Texture* GetTexture(const std::string& filename);` This function checks if the texture was not loaded before and if not, it creates and loads the Texture. It returns a pointer to the texture.

The class needs to keep track of all the filenames and their corresponding texture object pointers. A map container seems to be the perfect container choice here. Choose which map container would be most performant in this class and implement the class.

Add a TextureManager object to the game class. Add code that uses the object. You could also take the Smiley project, your game project, the minigame or another project to test the manager.

The same principle can be used to manage other resources, such as sounds.

4. Submission instructions

You have to upload the folder *1DAExx_10_name_firstname.* This folder contains 2 solution folders:

- W10
- Name_firstname_GameName which also contains your up-to-date report file.

Don't forget to clean up the solution before closing Visual Studio. And then:

- Remove the hidden folder .vs
- Remove the Debug folders

Remove the x64 folders, however not the x64 folders located in the Libraries sub folders.

5. References

5.1. The std::pair struct template

http://en.cppreference.com/w/cpp/utility/pair

5.2. minmax and minmax_element

http://en.cppreference.com/w/cpp/algorithm/minmax

http://en.cppreference.com/w/cpp/algorithm/minmax_element

5.3. Ordered containers

5.3.1. set

http://en.cppreference.com/w/cpp/container/set

5.3.2. multiset

http://en.cppreference.com/w/cpp/container/multiset

5.3.3. map

http://en.cppreference.com/w/cpp/container/map

5.3.4. multimap

http://en.cppreference.com/w/cpp/container/multimap

5.4. Unordered containers

5.4.1. unordered_set

http://en.cppreference.com/w/cpp/container/unordered_set

5.4.2. unordered_multiset

http://en.cppreference.com/w/cpp/container/unordered_multiset

5.4.3. unordered_map

http://en.cppreference.com/w/cpp/container/unordered_map

5.4.4. unordered_multimap

http://en.cppreference.com/w/cpp/container/unordered_multimap