Menu

My projects
Holy Graph
List projects
Available Cursus

Your projects

cub3d
Exam Rank 04
NetPractice

SCALE FOR PROJECT FT_CONTAINERS

You should evaluate 1 student in this team

Git repository

```
git@vogsphere-v2-bg.1
```

Introduction

Please comply with the following rules:

- Remain polite, courteous, respectful and constructive throughout the evaluation process. The well-being of the community depends on it.

- Identify with the student or group whose work is evaluated the possible dysfunctions in their project. Take the time to discuss and debate the problems that may have been identified.

- You must consider that there might be some differences in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade them as honestly as possible. The pedagogy is useful only and only if the peer-evaluation is done seriously.

Guidelines

- Only grade the work that was turned in the Git repository of the evaluated student or group.

- Double-check that the Git repository belongs to the student(s). Ensure that the project is the one expected. Also, check that 'git clone' is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you and make you evaluate something that is not the content of the official repository.

- To avoid any surprises and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).

- If you have not completed the assignment you are going to evaluate, you have to read the entire subject prior to starting the evaluation process.

- Use the available flags to report an empty repository, a non-functioning program, a Norm error, cheating, and so forth.
In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except for cheating, student are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated in the future.

- You should never have to edit any file except the configuration file if it exists. If you want to edit a file, take the time to explicit the reasons with the evaluated student and make sure both of you are okay with this.

- You must also verify the absence of memory leaks. Any memory allocated on the heap must be properly freed before the end of execution.
You are allowed to use any of the different tools available on the computer, such as leaks, valgrind, or e_fence. In case of memory leaks, tick the appropriate flag.

Attachments

subject.pdf main.cpp

Preliminary tests

If cheating is suspected, the evaluation stops here. Use the "Cheat" flag to report it. Take this decision calmly, wisely, and please, use this button with caution.

Prerequisites

The code must compile with c++ and the flags -Wall -Wextra -Werror

The code must compile with c++ and the flags -Wall -Wextra -Werror.

There is a Makefile, the project compiles correctly with the required options, is written in C++.

The code should also compile with the flag -std=c++98

Don't forget this project has to follow the C++98 standard. Thus, C++11 (and later) functions or containers are NOT expected.

Any of these means you must not grade the exercise in question:
- A function is implemented in a header file (except for template functions).
- A Makefile compiles without the required flags and/or another compiler than c++.

Any of these means that you must flag the project with "Forbidden Function":
- Use of a "C" function (*alloc, *printf, free).
- Use of a function not allowed in the exercise guidelines.
- Use of "using namespace " or the "friend" keyword (in this project, the "friend" keyword is allowed for specific uses, you'll see).
- Use of an external library, or features from versions other than C++98.
- Use of a STL container to implement another container.

Yes | No

Mandatory part

Verify that each mandatory container is correctly implemented. If the vector or the map is missing, check the 'Incomplete work' flag. But note the project can be passed without the stack. Do not hesitate to ask to the evaluated student what is the reference they used. A main.cpp is available to download on this page. It should compile with the student's containers. The evaluated student has also to turn in their own tests.

Vector - Basics

Make sure that every member function, non-member function and overload is present and works as expected. Same thing for iterators.
Use the STL container to check that everything works the same way.

Yes | No

Vector - Advance

- The inner data structure should be a dynamic array.
- const_iterator and iterators should be comparable.
- Check the dynamic reallocation system.
- Test the swap() function:
After swap, all the iterators, pointers and references pointing to elements in both containers remain valid and are now pointing to the same elements than before the call but in the other container (where they now iterate).
- Check that the friend keyword is used only for the relational operators.

Yes | No

Vector - Performance

Make sure that the speed is reasonable compared to the STL container!
For example, a deep copy should allocate all the memory in one call.

Yes | No

Map - Basics

Make sure that every member function, non-member function and overload is present and works as expected. Same thing for iterators.
Use the STL container to check that everything works the same way.

Yes | No

Map - Advance

- Check the inner structure. It should be an ordered tree (AVL tree, Red-Black tree, and so forth). For example, a simple array is not ok.
- Verify that pair<> is recoded and used.
- Verify that ft::make_pair() behaves as expected.

- There mustn't be several identical keys. Each key must be unique in map.
- Check that the keys are sorted using the comparison function (see Compare).
- Check std::allocator and allocator::rebind are used and there's no direct usage of new (rebind could be used in the tree).
- Check that insertion and deletion (erase) do not invalidate iterators.
- The swap() function should not move data but only pointers.
- Check that the friend keyword is used only for the relational operators and map::value_compare.
- There's no memory leak.

Yes | No

Map - Performance

Make sure that the speed is reasonable compared to the STL container!

Slower than the STL map is ok.
A complete timeout is not.
If it's more than 20 times slower than the map STL, count it false.

Yes | No

Stack - Basics

Make sure that every member function, non-member function and overload is present and works as expected.
Use the STL container to check that everything works the same way.

Yes | No

Stack - Advance

- The default underlying container must be the vector of the student.
- The underlying container must be protected and not private!
- The standard containers vector, deque and list are compatible as underlying containers.
- The stack cannot be iterate.

Yes | No

Bonus part

Evaluate the bonus part if, and only if, the mandatory part has been entirely and perfectly done, and the error management handles unexpected or bad usage. In case all the mandatory points were not passed during the defense, bonus points must be totally ignored.

Set

Make sure that every member function, non-member function and overload is present and works as expected. Same thing for iterators.
Use the STL container to check that everything works the same way.

The inner data structure must be a Red-Black tree.

Ask the evaluated student for details. How does it work? If you have any doubt, don't validate this bonus as the set is easier than the map.

The only bonus here is for the Red-Black tree!

Yes | No

Ratings

Don't forget to check the flag corresponding to the defense

Ok | Outstanding project

Empty work | Incomplete work | Invalid compilation | Cheat | Crash

Concerning situation | Leaks | Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation