Nexthink C++ Quiz

# Thank you very much for filling out this quiz to the best of your competences. The purpose of this quiz is to evaluate your level of expertise in technologies and techniques that we use daily @Nexthink. Good luck and thank you for your time!

Note: Feel free to skip exercises which do not lie in your area of competence, though we encourage you to still give them a go! Simply mention that it’s new material for you and we’ll take that into account.

1. Question

Given the code snippet:

int main() {

A a;

a.foo();

a.bar();

A\* b(new B());

b->foo();

b->bar();

delete b;

return 0;

}

And the corresponding program output:

A::A()

A::foo()

A::bar()

A::A()

B::B()

B::foo()

A::bar()

A::~A()

A::~A()

Please write classes A and B and describe why A is broken.

1. Question

A producer thread periodically produces elements and puts them in a queue, and a consumer thread takes the elements from the queue and does some processing with them.

Given the following class signature:

template<typename T>

class SyncQueue {

...

// Pops an element from the queue. It blocks if the queue is empty.

T pop();

// Pushes an element into the queue

Void push(const T& item);

...

};

Please write the implementation of the SyncQueue class using modern C++ synchronization primitives. What are the type requirements on T?

1. Question

Please add a move constructor to SyncQueue with an example how to use it. Could you please describe a few cases when the move semantic gives significant improvements for performance?

1. Question

Please discuss in general terms how you would make a piece of code scalable and fast? In particular, how do immutable and lock-free data structures help, how shared\_ptr may hinder performance and what are the issues with computation on multiple cores/processors?

1. Question

What design pattern can be used to represent tree structures?

1. Question

Please refactor this code to produce a better code with similar result. You are encouraged to use C++20 features whenever it makes sense. You may keep the current structure or completely redo the solution.

#include "stdlib.h"

#include "stdio.h"

#include "string.h"

#include "fcntl.h"

#define INVALID\_OPERATION (int)0

#define INVALID\_EXPRESION (int)1

#define DIVIDED\_BY\_ZERO (int)2

**struct** Expression **{**

Expression**(int** left**,** **const** **char\*** operation**,** **int** right**)** **{**

**if** **(**strcmp**(**operation**,** "+"**)** **&&** strcmp**(**operation**,**"-"**)** **&&**

strcmp**(**operation**,**"\*"**)** **&&** strcmp**(**operation**,**"/"**))**

**throw** INVALID\_OPERATION**;**

done **=** **false;**

value**[**0**]** **=** left**;**

value**[**1**]** **=** right**;**

expression**[**0**]** **=** NULL**;**

expression**[**1**]** **=** NULL**;**

**this->**operation **=** operation**;**

**}**

Expression**(**Expression**\*** left**,** **const** **char\*** operation**,** Expression**\*** right**)** **{**

**if** **(**strcmp**(**operation**,** "+"**)** **&&** strcmp**(**operation**,**"-"**)** **&&**

strcmp**(**operation**,**"\*"**)** **&&** strcmp**(**operation**,**"/"**))**

**throw** INVALID\_OPERATION**;**

**if** **(!**left **||** **!**right**)**

**throw** INVALID\_EXPRESION**;**

done **=** **false;**

expression**[**0**]** **=** left**;**

expression**[**1**]** **=** right**;**

**this->**operation **=** operation**;**

**}**

Expression**(int** left**)** **{**

**static** **const** **char\*** no\_op **=** ""**;**

done **=** **false;**

expression**[**0**]** **=** NULL**;**

expression**[**1**]** **=** NULL**;**

value**[**0**]** **=** left**;**

operation **=** no\_op**;**

**}**

**~**Expression**()** **{**

**if** **(**expression**[**0**])** **{**

**delete** expression**[**0**];**

**delete** expression**[**1**];**

**}**

**}**

**int** evaluate**()** **{**

**if** **(!**done**)** **{**

done **=** **true;**

**if** **(**strlen**(**operation**))** **{**

**if** **(!**strcmp**(**operation**,**"+"**))**

**if** **(**expression**[**0**])**

res **=** expression**[**0**]->**evaluate**()** **+** expression**[**1**]->**evaluate**();**

**else**

res **=** value**[**0**]** **+** value**[**1**];**

**else** **if** **(!**strcmp**(**operation**,** "-"**))**

**if** **(**expression**[**0**])**

res **=** expression**[**0**]->**evaluate**()** **-** expression**[**1**]->**evaluate**();**

**else**

res **=** value**[**0**]** **-** value**[**1**];**

**else** **if** **(!**strcmp**(**operation**,**"\*"**))**

**if** **(**expression**[**0**])**

res **=** expression**[**0**]->**evaluate**()** **\*** expression**[**1**]->**evaluate**();**

**else**

res **=** value**[**0**]** **\*** value**[**1**];**

**else** **if** **(!**strcmp**(**operation**,**"/"**))**

**if** **(**expression**[**0**])** **{**

**if** **(**expression**[**1**]->**evaluate**())**

res **=** expression**[**0**]->**evaluate**()** **/** expression**[**1**]->**evaluate**();**

**else**

**throw** DIVIDED\_BY\_ZERO**;**

**}** **else** **{**

**if** **(**value**[**1**])**

res **=** value**[**0**]** **/** value**[**1**];**

**else**

**throw** DIVIDED\_BY\_ZERO**;**

**}**

**}** **else**

res **=** value**[**0**];**

**}**

**return** res**;**

**}**

**void** dump**(**FILE**\*** fd**)** **{**

**if** **(**strlen**(**operation**))** **{**

**if** **(**expression**[**0**])** **{**

fprintf**(**fd**,** "("**);**

expression**[**0**]->**dump**(**fd**);**

fprintf**(**fd**,** ")"**);**

**}** **else**

fprintf**(**fd**,** "%d"**,** value**[**0**]);**

fprintf**(**fd**,** "%s"**,** operation**);**

**if** **(**expression**[**1**])** **{**

fprintf**(**fd**,** "("**);**

expression**[**1**]->**dump**(**fd**);**

fprintf**(**fd**,** ")"**);**

**}** **else**

fprintf**(**fd**,** "%d"**,** value**[**1**]);**

**}** **else** **{**

fprintf**(**fd**,** "%d"**,** value**[**0**]);**

**}**

**}**

**int** value**[**2**];**

Expression**\*** expression**[**2**];**

**const** **char\*** operation**;**

**bool** done**;**

**int** res**;**

**};**

Expression**\*** copy**(**Expression**\*** expression**)** **{**

**if** **(**strlen**(**expression**->**operation**))** **{**

**if** **(**expression**->**expression**[**0**])** **{**

**return** **new** Expression**(**copy**(**expression**->**expression**[**0**]),**

expression**->**operation**,** copy**(**expression**->**expression**[**1**]));**

**}** **else** **{**

**return** **new** Expression**(**expression**->**value**[**0**],**

expression**->**operation**,** expression**->**value**[**1**]);**

**}**

**}** **else** **{**

**return** **new** Expression**(**expression**->**value**[**0**]);**

**}**

**}**

**int** main**(int** argc**,** **char\*** argv**[])** **{**

Expression e1**(new** Expression**(**20**,** "."**,** 8**),** "/"**,** **new** Expression**(**6**));**

printf**(**"%d\n"**,** e1**.**evaluate**());**

Expression e2**(new** Expression**(**20**,** "-"**,** 8**),** "/"**,** **new** Expression**(**0**));**

**try** **{**

printf**(**"%d\n"**,** e2**.**evaluate**());**

**}** **catch** **(int** i**)** **{**

printf**(**"nan\n"**);**

**}**

Expression**\*** e3**(**copy**(&**e1**));**

printf**(**"%d\n"**,** e3**->**evaluate**());**

**delete** e3**;**

e1**.**dump**(**fopen**(**"expression.txt"**,** "w"**));**

**return** 0**;**

**}**

1. Question

Why is UNIX signal handling so tricky, especially in a multithreaded application?

1. Question

Could you describe how would you implement an asynchronous IO dispatcher, which is gentle on process resources? You can use select, poll or something else on your choice.