Demo exam tasks

Åsmund Eldhuset DAVE3606 at OsloMet, spring 2025

These tasks are only intended for general exam practice, and to demonstrate the instructor's general task style. The total amount of work is *not* intended to be representative of the amount of work on the actual exam. Please do *not* try to use this to guess what will be on the exam — a few of these tasks overlap partly with the actual exam tasks, some share similar themes but do not directly overlap, and some are not at all related to any exam task.

Intro (Lecture 1)

1. Which of the following is generally the most important factor for how fast your program will run (given a specific input)?
    - ○ Your choice of programming language
    - ○ Your choice of algorithm
    - ○ Whether you write `i++` or `++i` in your loops
    - ○ The size of your compiled binary
    - ○ The CPU speed of your computer
    - ○ The amount of memory on your computer
    - ○ The size of your computer's hard disk

Analysis (Lectures 2 and 3)

2. Consider the following code. `contains_duplicates` will be called with a list that contains n integers, where the absolute value of each integer is less than 2^{31}. Express the worst-case runtime complexity and best-case runtime complexity of `contains_duplicates`, using the Θ notation for both. Then, express the runtime complexity of the algorithm as a whole, using the O and Ω notations. Why would it not be appropriate to use the Θ notation for that?

```python
def contains(values, value_to_find, ignore_index):
    for i, v in enumerate(values):
        if i != ignore_index and v == value_to_find:
            return True
    return False

def contains_duplicates(values):
    for i, v in enumerate(values):
        if contains(values, v, i):
            return True
    return False
```

   (Note that what `for i, v in enumerate(values)` does is to loop over the elements `v` in the list while also telling you the index `i` of each element.)

3. Consider some algorithm that, when it receives n input elements, performs $7n^2 + 3n + 9$ steps in the worst case and $8n + 4$ steps in the best case. Which of the following are correct ways to describe the algorithm's overall runtime complexity?
    - ☐ $\Omega(1)$
    - ☐ $\Omega(n)$
    - ☐ $\Omega(n^2)$
    - ☐ $\Omega(n^3)$
    - ☐ $\Theta(1)$

☐ $\Theta(n)$

☐ $\Theta(n^2)$

☐ $\Theta(n^3)$

☐ $O(1)$

☐ $O(n)$

☐ $O(n^2)$

☐ $O(n^3)$

Memory (Lectures 4 and 5)

4. The primitive data types in Java are (alphabetically): `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, and `short`. Fill them into this table based on what type of data they contain and how large they are.

Byte size	Integers	Floating-point	Others
8			
4			
2			
1			

5. Suppose that we declare `Person* p = nullptr;` in a C++ program. Which of the following code lines exhibit undefined behavior (most likely crashing the program or producing incorrect results)? (The code lines are not meant to all be a part of one program; think of each line independently of the others, following the declaration with just that one line.)

☐
```
if (p == nullptr) { ... }
```

☐
```
if (p != nullptr) { ... }
```

☐
```
cout << p << endl;
```

☐
```
cout << *p << endl;
```

☐
```
cout << p->name << endl;
```

☐
```
cout << p->getAge() << endl;
```

☐
```
Person* q = p;
```

☐
```
p = new Person("Siri", 47);
```

☐
```
p = nullptr;
```

6. Consider the following recursive function:

```
1  void recurse(int x) {
2      cout << x << endl;
3      recurse(x + 1);
4  }
5
6  int main() {
7      recurse(0);
8      return 0;
9  }
```

What will happen when we run this program? Briefly explain why.

- ○ The program will immediately crash
- ○ The program will print 0 and then exit successfully
- ○ The program will loop infinitely, printing 0, 1, 2, 3, ...
- ○ The program will crash after printing a few thousand numbers
- ○ The program will crash after printing a few billion (1 000 000 000) numbers

File I/O (Lectures 6 and 7)

Using files

7. When a program opens a file, the operating system gives the file a file descriptor. What is it important to do with this file descriptor when you no longer need the file?
8. What is the purpose of the operating system's disk cache and the disk buffer offered by your programming language? Describe one pattern of reading data that would be very slow without any of these.

Encoding

9. Which of the following statements about character sets and encodings are true?

   ☐ UTF-8 comes in both a big-endian and a little-endian version.

   ☐ UTF-16 comes in both a big-endian and a little-endian version.

   ☐ UTF-32 comes in both a big-endian and a little-endian version.

   ☐ Text in any UTF encoding may begin with a special character that indicates which encoding it is and (if relevant) which endianness is used.

   ☐ The UTF-8 encoding is *self-synchronizing* (even if you have lost some bytes when transmitting data, you can tell from looking at a byte whether it is the start of a character or the "inside" of a character).

   ☐ The first byte of a character in the UTF-8 encoding indicates how many bytes that character consists of.

10. The byte sequence `C3 86 52 41` (written in hexadecimal) represents the UTF-8 encoding of some text. What is the text? (Use the table below.)

    Hint: The Unicode code point for capital A is 65 (in decimal), B is 66, and so on, up to Z, which is 90; Æ is 198, Ø is 216, and Å is 197.

Antall bits	Bitmønster	UTF-8-enkoding
1–7	aaaaaaa	0aaaaaaa
8–11	bbbbbaaaaaa	110bbbbb 10aaaaaa
12–16	ccccbbbbbbaaaaaa	1110cccc 10bbbbbb 10aaaaaa
17–21	dddcccccbbbbbbaaaaaa	11110ddd 10cccccc 10bbbbbb 10aaaaaa

File formats

11. What is the difference between *lossless* and *lossy* compression?
12. The CSV file format uses commas and newlines as special characters, which denote separation between cells and lines. How does the format deal with values that contain these characters? Does the format's approach to that problem require even more characters to be considered special, and how are *they* in turn handled?

Cross-language interoperation (Lecture 8)

13. According to Linux programming conventions, what is the significance of the status code (also called return code or exit code) of a process?

Templates and generics (Lectures 9 and 10)

14. Write a static Java function `withoutNulls` with one generic type parameter `T`, which takes a parameter that is a list of `T` and returns those elements that are not `null` (also as a list of `T`). The list that is passed to the function should not be modified.

    Then, write a similar function in C++.

    Imagine that you write one Java program and one C++ program, and in both programs, you use these functions with many different types. With which language would you expect the size of the compiled binary to be the largest? Why?

    What would happen if we were to use the C++ function with a non-pointer type, where the elements cannot possibly be null? Will it compile or not? (Recall that on the real exam, you won't have the compiler available to tell you!)

Design patterns (Lectures 11 and 13)

15. Here is a Java interface and a class that does *not* implement the interface:

```java
public interface Vehicle {
    int getWheelCount();
}

public class Car {
    private int numberOfWheels;

    public Car(int numberOfWheels) {
        this.numberOfWheels = numberOfWheels;
    }

    public int getNumberOfWheels() {
        return numberOfWheels;
    }
}
```

    ("Vehicle" means "kjøretøy".) Assume that you do not own this code, so that you are *not* allowed to modify it. Use one of the design patterns from the course to make it possible to (indirectly) pass a `Car` to a function that takes a `Vehicle` as a parameter. What is the name of that pattern?

Caching (Lecture 13)

16. Assume that there is a function `slow_computation` that takes a 32-bit integer as a parameter and performs some slow computation. Someone has written a cache for it, and they're saying that a nice thing about their cache is that it will never contain more than 2^{16} elements, thus limiting memory usage over time.

```python
cache = {}

def cached_computation(n):
    cache_key = n % (2 ** 16)
    if cache_key in cache:
        return cache[cache_key]
    result = slow_computation(n)
    cache[cache_key] = result
    return result
```

    (`%` is the *modulo* operator, returning the remainder after division. `**` is the power operator.) While it is generally a good idea to limit the cache size, this is a very bad way of doing it. Why? What could go wrong?