



UNIVERSITETET I BERGEN

KANDIDAT

117

PRØVE

INF214 0 Multiprogrammering

Emnekode	INF214
Vurderingsform	Skriftlig eksamen
Starttid	29.02.2024 08:00
Sluttid	29.02.2024 11:00
Sensurfrist	--
PDF opprettet	13.11.2024 11:42

Exam structure

Oppgave	Tittel	Status	Poeng	Oppgavetype
i	Exam structure 29.2.2024			Informasjon eller ressurser

Theoretical questions

Oppgave	Tittel	Status	Poeng	Oppgavetype
1	Task 1	Besvart	10/10	Langsvar
2	Task 2	Riktig	4/4	Flervalg (flere svar)
3	Task 3	Besvart	7/7	Langsvar

Semaphores

Oppgave	Tittel	Status	Poeng	Oppgavetype
4	Task 4	Besvart	19/25	Programmering

Monitors

Oppgave	Tittel	Status	Poeng	Oppgavetype
5	Task 5	Riktig	11/11	Nedtrekk
6	Task 6	Riktig	8/8	Paring

CSP

Oppgave	Tittel	Status	Poeng	Oppgavetype
7	Task 7	Besvart	4/5	Programmering

JavaScript generators and iterators

Oppgave	Tittel	Status	Poeng	Oppgavetype
8	Task 8	Delvis riktig	2.5/5	Sammensatt
JavaScript promises				
Oppgave	Tittel	Status	Poeng	Oppgavetype
9	Task 9	Ubesvart	5/10	Langsvar
10	Task 10	Riktig	9/9	Fyll inn tekst
i	Cheat sheet about the semantics of promises			Informasjon eller ressurser
11	Task 11.1	Riktig	1/1	Flervalg
12	Task 11.2	Riktig	1/1	Flervalg
13	Task 11.3	Riktig	1/1	Flervalg
14	Task 11.4	Riktig	1/1	Flervalg
15	Task 12	Riktig	2/2	Paring

1 Task 1

Consider the following program:

```
int x = 1;
int y = 4;
co
  < x = x * y; >
||
  x = y - x;
oc
```

Answer the following questions:

- Does the program meet the requirements of the At-Most-Once-Property? Explain your answer.
- What are the possible final values of x and y? Explain your answer.

Fill in your answer here

For simplicity

<x = x * y>: is called A

x = y - x : is called B

1)

No, this program does not meet the requirements for At-Most-Once-Property. Since in line B, the process makes a critical reference to the variable x and this variable is also written to by another process (in A). Therefore, the value that x will have after the termination of the program is not deterministic and is dependent on the order and length of time each of these process run for.

2)

There are three different final values for x and y (I say, but the value of y is never changed)

1. x = 0, y = 4

- Here, both process run in order A then B to their full completion. First A runs to completion, then B runs to completion

2. x = 12, y = 4

- Here, both processes also run to completion once started, but first B, then A.

3. x = 3, y = 4

- Here, process B reads the value of 'x', but then its execution is stopped and the process A starts and fulfills its execution (since it is atomic) then process B resumes and finishes its execution with the original value of x that it read before being stopped.

Ord: 211

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

0988353

Fyll inn oppgavekode og emneinformasjon på alle skissearkene

Fill out question code and test information on every sheet

Oppgavekode
Question codeDato
DateEmnekode
Subject codeKandidatnummer
Candidate numberOppgavenummer
Question numberSidetall
Page number

0	9	8	8	3	5	3
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6
7	7	7	7	7	7	7
8	8	8	8	8	8	8
9	9	9	9	9	9	9

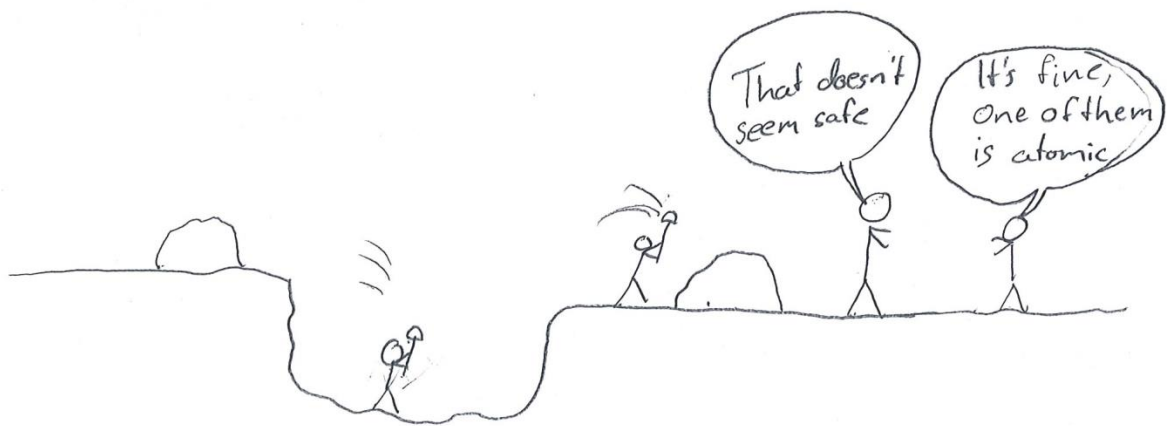
29.02.24 INF 214

117

1

3 av/of 3

Tegneområde Drawing area



Not part
of answer!

2 Task 2

Consider the following program written in the AWAIT language:

```
bool can_proceed = false;
bool should_continue = true;
co
    while (should_continue) {
        can_proceed = true;
        can_proceed = false;
    }
||
    <await (can_proceed) should_continue = false;>
oc
```

Which of the following statements hold for this program?

Select one or more alternatives:

☒ If the scheduling policy is strongly fair, this program will eventually terminate



☐ If the scheduling policy is strongly fair, this program will never terminate

☒ If the scheduling policy is weakly fair, this program might not terminate



☐ If the scheduling policy is weakly fair, this program will never terminate

☐ No matter what the scheduling policy is, this program will never terminate

☐ No matter what the scheduling policy is, this program will always terminate

Maks poeng: 4

Knytte håndtegninger til denne
oppgaven?
Bruk følgende kode:

2 2 5 5 7 3 5

3 Task 3

Summarize very briefly the purpose of **barriers**.

Fill in your answer here

Barrier synchronisation is used to make sure that some processes won't race to far ahead of other processes. We instead can specify that all processes should wait at a barrier until some criteria is fulfilled. Example, some process might do work that is required for other ones that are ahead.

Ord: 50

Maks poeng: 7

Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

5491996

Fyll inn oppgavekode og emneinformasjon på alle skissearkene

Fill out question code and test information on every sheet

Oppgavekode
Question codeDato
DateEmnekode
Subject codeKandidatnummer
Candidate numberOppgavenummer
Question numberSidetall
Page number

5	4	9	1	9	9	6
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6
7	7	7	7	7	7	7
8	8	8	8	8	8	8
9	9	9	9	9	9	9

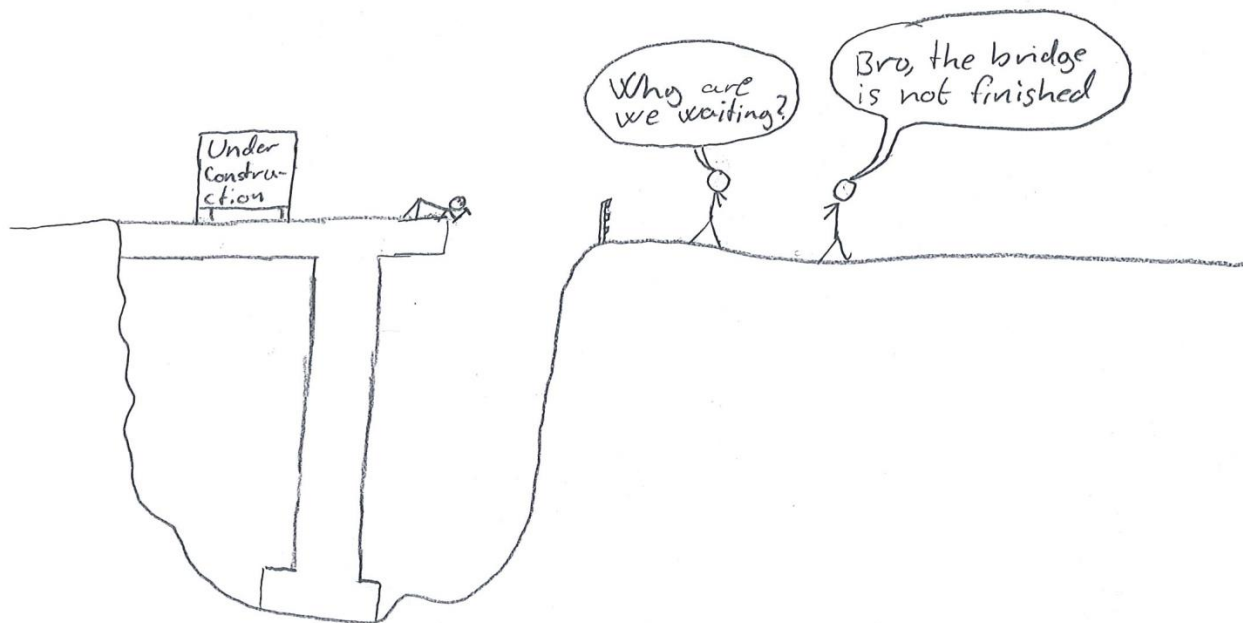
29.02.24 INF 214

117

3

2 av/of 3

Tegneområde Drawing area



Not real part
of answer!

4 Task 4

Person *A* invited friends *B*, *C*, and *D* to cook some pasta together. To make a portion of pasta, three ingredients are needed: noodles, cheese, sauce.

Each of these persons *B*, *C*, *D* has only one type of each of the ingredients: person *B* has noodles, person *C* has cheese, and person *D* has sauce.

We assume that persons *B*, *C*, and *D* each has an unlimited supply of these ingredients (i.e., of noodles, of cheese, of sauce), respectively. Person *A*, who invited them, also has an unlimited supply of all the ingredients.

Here is what happens: person *A* puts two random ingredients on the table. Then the invited person who has the third ingredient picks up these other two ingredients, and makes the pasta (i.e., takes a handful of noodles, adds some sauce, and puts on top some cheese), and then eats the pasta. Person *A* waits for that person to finish. This "cycle" of is then infinitely repeated.

Write code in the AWAIT language that simulates this situation.

- **Represent the persons *A*, *B*, *C*, *D* as processes.**
- **You must use a SPLIT BINARY SEMAPHORE for synchronization.**
- **Make sure that your solution avoids deadlock.**
- **EXPLAIN very briefly the advantages of using the split binary semaphore.**

Fill in your answer here

```

1  Ingridient ingridient1;
2  Ingridient ingridient2;
3
4  sem hostA = 1;
5  sem noodleBAccess = 0;
6  sem cheeseCAccess = 0;
7  sem sauceDAccess = 0;
8
9  process HasNoodles {
10     while(True) {
11         P(noodleBAccess);
12         makePasta(ingridient1, ingridient2);
13         eatPasta();
14         V(hostA);
15     }
16 }
17
18 process HasCheese {
19     while(True) {
20         P(cheeseCAccess);
21         makePasta(ingridient1, ingridient2);
22         eatPasta();
23         V(hostA);
24     }
25 }
26
27 process HasSauce {
28     while(True) {
29         P(sauceDAccess);
30         makePasta(ingridient1, ingridient2);
31         eatPasta();
32         V(hostA);
33     }
34 }
35
36 process Host {
37     while(True) {

```

```
37 while(True) {
38     P(HostA);
39     ingridient1 = chooseRandomIngridient();
40     ingridient2 = chooseRandomIngridient(ingridient1); //Excluding ingridient1
41
42     if ((ingridient1 == cheese AND ingridient2 == sauce) OR (ingridient1 == sauce
43         V(noodleBAccess);
44
45     } else if ((ingridient1 == noodles AND ingridient2 == sauce) OR (ingridient1
46         {
47         V(cheeseCAccess);
48     } else if ((ingridient1 == noodles AND ingridient2 == cheese) OR (ingridient
49         )) {
50         V(sauceDAccess);
51     }
52 }
53 // I feel like this is a pretty bad party where you have to look at someone make pas
54 // And to make it even worse, you might never get your turn to make and eat pasta ev
55 // Rating: Breaking physics was fun, but I want to leave, 7/5 would do again. :b
```

Maks poeng: 25

**Knytte håndtegninger til denne
oppgaven?**
Bruk følgende kode:

8 7 3 9 6 8 1

5 Task 5

Consider the following variant of the Readers/Writers problem: reader processes query a database and writer processes examine and modify it. Readers may access the database concurrently, but writers require exclusive access. Although the database is shared, we cannot encapsulate it by a monitor, because readers could not then access it concurrently since all code within a monitor executes with mutual exclusion. Instead, we use a monitor merely to arbitrate access to the database. The database itself is global to the readers and writers.

The *arbitration monitor* grants permission to access the database. To do so, it requires that processes inform it when they want access and when they have finished. There are two kinds of processes and two actions per process, so the monitor has four procedures:

- **request_read**,
- **request_write**,
- **release_read**,
- **release_write**.

These procedures are used in the obvious ways. For example, a reader calls **request_read** before reading the database and calls **release_read** after reading the database.

To synchronize access to the database, we need to record how many processes are reading and how many processes are writing. In the implementation below, **nr** is the number of readers, and **nw** is the number of writers; both of them are initially 0. Each variable is incremented in the appropriate request procedure and decremented in the appropriate release procedure.

A software developer has started on the implementation of this monitor. Your task is to fill in the missing parts.

Your solution does not need to arbitrate between readers and writers.

```
monitor RW_Controller {
  int nr = 0;
  int nw = 0;
  cond OK_to_write; // signalled when nr == 0 and nw == 0
  cond OK_to_read; // signalled when nw == 0
```

```
// reading:
```

```
procedure request_read() {
```

```
  while (nw>0)
```



```
  (while (nw>0), while (nr>0), while (nr>0 || nw>0), if (nr>0), while
(nr<0), /* conditional or loop is not needed here */, if (nr>0 || nw>0), if (nw>0), if (nw<0), while
(nw<0), if (nr==0), while (nr<0 || nw<0), while (nw==0), if (nr<0 || nw<0), while (nr==0), if (nw==0),
if (nr<0)) {
```

```
    wait(OK_to_read)
```



```
    (signal(OK_to_read), // nothing is needed here,
wait(OK_to_read), signal(OK_to_write), signal_all(OK_to_write), wait(OK_to_write),
signal_all(OK_to_read))
```

```
};
nr = nr + 1;
```

```
// nothing is needed here ✓ (signal(OK_to_write), wait(OK_to_read), signal(OK_to_read),
wait(OK_to_write), signal_all(OK_to_read), // nothing is needed here, signal_all(OK_to_write))
}
```

```
procedure release_read() {
```

```
// nothing is needed here ✓ (signal_all(OK_to_write), signal(OK_to_read),
signal(OK_to_write), // nothing is needed here, wait(OK_to_write), wait(OK_to_read),
signal_all(OK_to_read))
nr = nr - 1;
```

```
if (nr==0) ✓ (while (nw>0), if (nr<0), while (nw<0), if (nr>0 || nw>0), while (nr<0 ||
nw<0), while (nr<0), while (nw==0), if (nw>0), /* conditional or loop is not needed here */, if
(nr==0), if (nr>0), while (nr>0), if (nw<0), if (nr<0 || nw<0), while (nr==0), if (nw==0), while (nr>0 ||
nw>0)) {
```

```
signal(OK_to_write) ✓ (signal_all(OK_to_read), signal(OK_to_write),
wait(OK_to_read), signal_all(OK_to_write), // nothing is needed here, signal(OK_to_read),
wait(OK_to_write))
}
}
```

```
// writing-related:
```

```
procedure request_write() {
```

```
while (nr>0 || nw>0) ✓ (while (nr<0 || nw<0), if (nw==0), if (nr>0), if (nr<0 || nw<0), if
(nw<0), if (nw>0), while (nw>0), while (nr==0), while (nr>0), while (nw==0), if (nr>0 || nw>0), if
(nr==0), while (nr<0), while (nr>0 || nw>0), while (nw<0), if (nr<0), /* conditional or loop is not
needed here */)
}
```

```
wait(OK_to_write) ✓ (wait(OK_to_write), signal(OK_to_read), wait(OK_to_read), //
nothing is needed here, signal_all(OK_to_write), signal_all(OK_to_read), signal(OK_to_write))
}
nw = nw + 1;
```

```
// nothing is needed here ✓ (signal(OK_to_read), signal(OK_to_write),
signal_all(OK_to_read), wait(OK_to_read), signal_all(OK_to_write), wait(OK_to_write), // nothing is
```


needed here)

}

procedure release_write() {

// nothing is needed here



(wait(OK_to_read), signal_all(OK_to_write),

signal(OK_to_read), wait(OK_to_write), // nothing is needed here, signal_all(OK_to_read),

signal(OK_to_write))

nw = nw - 1;

signal(OK_to_write);

signal_all(OK_to_read)



(wait(OK_to_write), wait(OK_to_read), signal(OK_to_read), //

nothing is needed here, signal_all(OK_to_write), signal(OK_to_write), signal_all(OK_to_read))

}

}

Maks poeng: 11

**Knytte håndtegninger til denne
oppgaven?**

Bruk følgende kode:

8 4 3 4 0 3 4

6 Task 6

There is a duality between monitors and message passing. What is that duality exactly?

In the table, the rows represent notions about monitors, and the columns represent notions about message passing.

Click the circle in a cell to represent that a notion about monitors is dual to a notion about message passing.

Please match the values:

	save pending request	`send reply()`	arms of case statement on operation kind	`send request(); receive reply()`	retrieve and process pending request	`request` channel and operation kinds	`receive request()`
procedure bodies	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
`signal`	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
`wait`	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
procedure call	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
monitor entry	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
permanent variables	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
procedure return	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
procedure identifiers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Maks poeng: 8

Knytte håndtegninger til denne oppgaven?

9992847

Bruk følgende kode:

7 Task 7

Using Communicating Sequential Processes, define a process **Copy** that copies a character from process **Bergen** to process **Vestland**.

Fill in your answer here

```
1 Process Copy {  
2   Char c;  
3   True ->  
4   do Bergen?c -> Vestland!c od  
5 }
```

Maks poeng: 5

Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

4 1 6 9 2 7 8

8 Task 8

What will be printed when the following JavaScript code is executed?

```
function* foo(x) {  
  var y = x + (yield true);  
  return y;  
}  
var it = foo(10);  
var res = it.next(20);
```

console.log(res.value); // what will be printed here? Answer:



res = it.next(30);

console.log(res.value); // what will be printed here? Answer:



(40)

Maks poeng: 5

Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

5 7 8 1 3 4 8

9 Task 9

<i>line number</i>	
1	var a = promisify({});
2	var b = promisify({});
3	var c = b.onReject(x => x - 1);
4	a.link(b);
5	a.reject(1);

Consider the JavaScript code on the image.

Note the syntax here is a blend of JavaScript and λ_p , which uses:

- **promisify** to create a promise,
- **onReject** to register a reject reaction,
- **link** to link to promises (linking means that when the original promise is resolved/rejected, then the linked promise will be resolved/rejected with the same value)

Draw a promise graph for this code.

Remember to use the names of nodes in that graph that represent the "type" of node:

- v** for value
- f** for function
- p** for promise

with a subscript that represents the **line number** where that particular value/function/promise has been **declared / where it appears first**.

For example, the integer value 1 on line 5 will be denoted by **v₅** in the promise graph.

Please draw the promise graph on a piece of Scantron paper that you will get during the exam.

If you did not receive this, please request this now. Read the instructions on the Scantron paper carefully to ensure your answer on paper will be uploaded correctly after the exam. You will need to hand in this paper after the exam.

DO NOT WRITE ANYTHING IN THIS TEXT FIELD.

Ord: 0

**Knytte håndtegninger til denne
oppgaven?**
Bruk følgende kode:

3 9 7 3 3 2 1

Fyll inn oppgavekode og emneinformasjon på alle skissearkene

Fill out question code and test information on every sheet

Oppgavekode
Question codeDato
DateEmnekode
Subject codeKandidatnummer
Candidate numberOppgavenummer
Question numberSidetall
Page number

3	9	7	3	3	2	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6
7	7	7	7	7	7	7
8	8	8	8	8	8	8
9	9	9	9	9	9	9

29.02.24

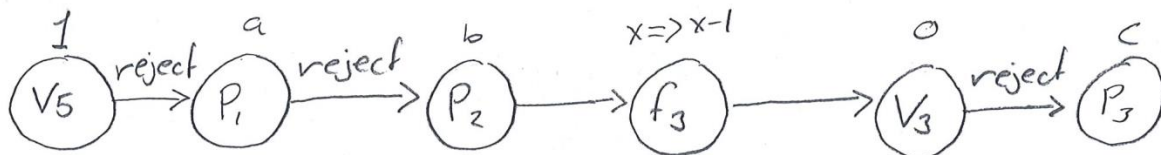
INF214

117

9

1 av/of 3

Tegneområde Drawing area


















10 Task 10

Consider the following HTML/JavaScript attached in the PDF file to this question.

- This user **never** clicks the button **myOtherButton**.
- The user clicks the button **myButton** 7 (seven) milliseconds after the execution starts.

What happens at particular time points?

Write an integer number in each of the text boxes. If something mentioned in the left column on the table does not happen, then write "-1" (negative one) in the corresponding right column.

what happens	at what time	
`clickHandler` starts	<input type="text" value="18"/>	 milliseconds
`clickHandler` finishes	<input type="text" value="28"/>	 milliseconds
interval fires (for the first time)	<input type="text" value="10"/>	 milliseconds
interval fires (for the second time)	<input type="text" value="20"/>	 milliseconds
interval fires (for the third time)	<input type="text" value="30"/>	 milliseconds
`intervalHandler` starts (for the first time)	<input type="text" value="38"/>	 milliseconds
`intervalHandler` finishes (for the first time)	<input type="text" value="46"/>	 milliseconds
mainline execution starts	0 milliseconds	
mainline execution finishes	18 milliseconds	
`otherClickHandler` starts	<input type="text" value="-1"/>	 milliseconds
`otherClickHandler` finishes	<input type="text" value="-1"/>	 milliseconds
promise handler starts	<input type="text" value="28"/>	 milliseconds
promise handler finishes	<input type="text" value="32"/>	 milliseconds
promise resolved	a tiny bit after <input type="text" value="18"/>	 milliseconds
`timeoutHandler` starts (for the first time)	<input type="text" value="32"/>	 milliseconds
`timeoutHandler` finishes (for the first time)	<input type="text" value="38"/>	 milliseconds
`timeoutHandler` starts (for the first time)	<input type="text" value="-1"/>	 milliseconds

what happens	at what time
--------------	--------------

`timeoutHandler` finishes (for the second time)	-1	✓	milliseconds
timer fires (for the first time)	10	✓	milliseconds
timer fires (for the second time)	-1	✓	milliseconds
user clicks the button	7 milliseconds		

Maks poeng: 9

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

7 8 4 2 0 2 5**11 Task 11.1**

There are four rules shown in the PDF attached to this question.

Which of the rules **registers a fulfill reaction on a pending promise?**

Answer:

- ☐ Rule A
- ☐ Rule B
- ☒ Rule C
- ☐ Rule D



Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:


6 1 5 8 1 3 0

12 Task 11.2

$$\frac{a \in Addr \quad a \in \text{dom}(\sigma) \quad \psi(a) \in \{F(v'), R(v')\}}{\langle \sigma, \psi, f, r, \pi, E[a.\text{resolve}(v)] \rangle \rightarrow \langle \sigma, \psi, f, r, \pi, E[\text{undef}] \rangle}$$

What does this rule describe?

Select one alternative:

- ☒ This rule states that resolving a settled promise has no effect. 
- ☐ This rule handles the case when a pending promise is resolved.
- ☐ This rule turns an address into a promise
- ☐ This rule handles the case when a fulfill reaction is registered on a promise that is already resolved.

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

3 2 2 5 0 5 8

13 Task 11.3

There are four rules shown in the PDF attached to this question.

Which of the rules **turns an address into a promise**?

Answer:

☐ Rule E

☒ Rule F



☐ Rule G

☐ Rule H

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:


4 3 8 5 7 8 8

14 Task 11.4

$$\begin{array}{c}
 a_1 \in Addr \quad a_1 \in \text{dom}(\sigma) \quad a_2 \in Addr \quad a_2 \in \text{dom}(\sigma) \quad \psi(a_1) = F(v) \\
 \pi' = \pi :: (F(v), \text{default}, a_2) \\
 \hline
 \langle \sigma, \psi, f, r, \pi, E[a_1.\text{link}(a_2)] \rangle \rightarrow \langle \sigma, \psi, f, r, \pi', E[\text{undef}] \rangle
 \end{array}$$

What does this rule describe?

Select one alternative:

- ☐ This rule causes a pending promise to be "linked" to another.
- ☐ This rule causes a promise to be "linked" to another, with no regards to the state of that original promise.
- ☒ This rule causes an already settled promise to be "linked" to another. 
- ☐ This rule causes a non-settled promise to be "linked" to another.

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
 Bruk følgende kode:

6 1 5 7 2 0 7

15 Task 12

What kind of bugs can be detected by what kind of situations in a promise graph?

In the table, the rows describe various conditions on nodes and edges in a promise graph, and the columns are names of bugs.

Please note that there are 4 rows and 5 columns in the table, so one of the columns is irrelevant.

Please match:

	another type of bug	Bug "Double Resolve/Reject"	Bug "Dead Promise"	Bug "Missing Exceptional Reject Reaction"	Bug "Missing Resolve/Reject Reaction"
a promise with a reject edge, but no reject registration edge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
a promise that has no outgoing registration edges	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
a promise with no resolve or reject edges nor any link edge	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
multiple resolve (or reject) edges leading to the same promise	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maks poeng: 2

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

5 8 3 4 1 6 2