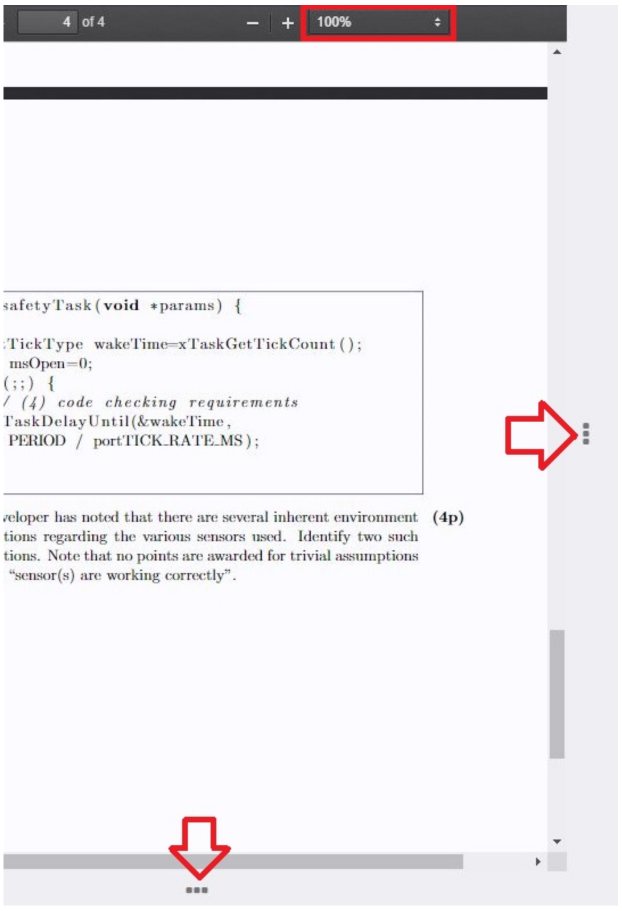
Exam Instructions

As communicated in a recent email, the exam is similar to a regular exam in terms of contents. What differs is how the exam is taken, namely:

- how each exercise shows up
- what are the input forms
- what are you allowed/not allowed to do.

With respect to the last item, you can browse the Internet/the course material, the only thing we ask is that you *do not collaborate with other students*. For the first two items, we use the features provided by Inspera.

Each exercise (of which there are 4) appears on the left of your screen, while on the right you will find answer form(s). On this page we include exercise 1, as an example, shown on the left panel. There is no answer form so you cannot give answers on this page. If you click on the exercise panel you should be able to scroll up and down from the beginning to the end of the exercise. In case the text on the panel appears illegible or you encounter formatting problems, disable auto-zoom and select a zoom option which gives you the best readability. Also useful are options for resizing the panel. These are highlighted by red arrows on the image below.



```
safetyTask(void *params) {

:TickType wakeTime=xTaskGetTickCount();
 msOpen=0;
(;;) {
/ (4) code checking requirements
TaskDelayUntil(&wakeTime,
 PERIOD / portTICK_RATE_MS);
```

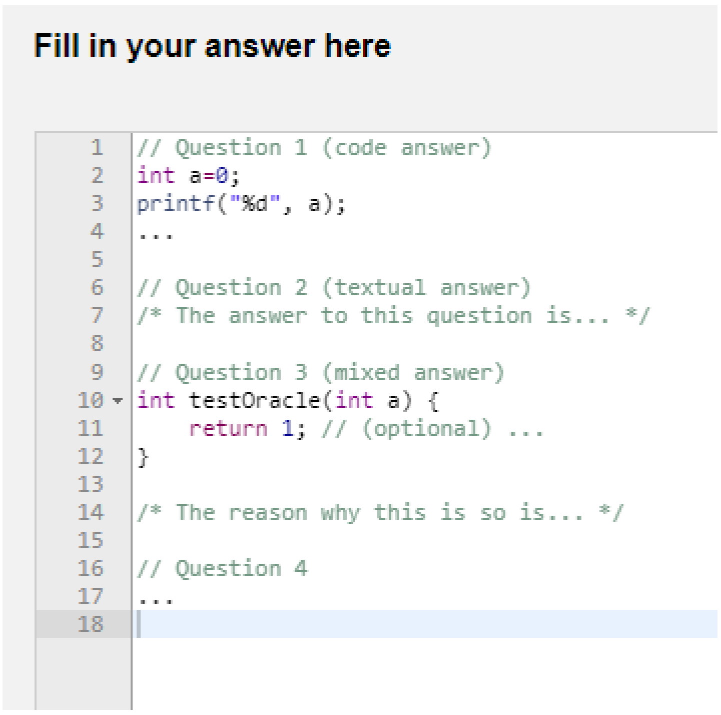
veloper has noted that there are several inherent environment (4p)
tions regarding the various sensors used. Identify two such
tions. Note that no points are awarded for trivial assumptions
"sensor(s) are working correctly".

Before proceeding any further, ensure you find the settings with which all the text/code of Exercise 1 is readable. On a small screen it is possible that you still encounter issues with scrolling. Hence we also make the exam full available for download as .pdf from the link below. In case you use this option, you can minimize the left panel displaying exercise.

Exam Download Link

Exercise 1 provides multiple-choice questions with single answers. For each question there is a corresponding form which allows you to choose exactly one answer. Old rules apply, a correct answer awards you 2 points, an incorrect answer losses you 1 point. If unsure, it is (often) better to leave the question unanswered.

Exercises 2 and 3 have specialized input forms which recognize the C programming language, providing syntax highlighting and automatic indentation. We thought using such forms made sense since most questions require writing C code. Unfortunately, limitations of the platform mean there is only one form per exercise. Hence, we expect you to deliver answers as shown in the example below.



Fill in your answer here

```
// Question 1 (code answer)
int a=0;
printf("%d", a);
...

// Question 2 (textual answer)
/* The answer to this question is... */

// Question 3 (mixed answer)
int testOracle(int a) {
    return 1; // (optional) ...
}

/* The reason why this is so is... */

// Question 4
...

```

Concretely, answers are clearly delimited by line comments containing the corresponding question number ("// Question 1 "). Code is written directly into the form. To help indentation, you may want to copy the context (e.g. the method signature, or the switch block) where the code is supposed to fit in. Explanations are provided inside block (/* multi-line explanation */) or line comments (// single-line explanation). If you think it enhances your answer, you may add comments to the code itself.

Exercise 4 provides a standard text input forms for each question.

At the end of the exam there is a feedback form which we kindly ask you to fill in. This will help use improve the experience next time.

Should you have doubts regarding questions, you can contact me at paul.fiterau_brostean@it.uu.se . I will try to reply within one hour.

Good luck!

1 Exercise 1: Multiple choice questions (20 points)

Select one answer (alternative) for each of the questions in Exercise 1. 2 points are given for each correct answer, 1 point is subtracted for each wrong answer (the total number of points achieved in this exercise is at least 0).

Question 1:
Select one alternative:

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 2:
Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 3:
Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 4:

Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 5:

Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 6:

Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 7:

Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 8:

Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3
- ○ Alternative 4

Question 9:
Select one alternative

- ○ Alternative 1
- ○ Alternative 2
- ○ Alternative 3

Question 10:
Select one alternative

- ○ Alternative 1
- ○ Alternative 2

Maximum marks: 20

2 Exercise 2: RTOS Programming and Concurrency (17 points)

Fill in your answer for Questions 1 through 3

Maximum marks: 17

3 Exercise 3: Testing and Coverage (17 points)

Fill in your answers for Questions 1 through 3.

Maximum marks: 17

4 Exercise 4: Arithmetic (6 points)

Fill in your answer for Question 1

Fill in your answer for Question 2

Maximum marks: 6

Feedback

Before handing in, we would appreciate if you spared a few minutes to give feedback so that we can improve the experience next time.

On what device did you take the exam?

- ○ Desktop
- ○ Laptop (>= 13')
- ○ Small Laptop (< 13')
- ○ Tablet
- ○ Smartphone

How would you rate your exam-taking experience?

- ○ Good
- ○ OK
- ○ Bad
- ○ Terrible

Where did you read the exercises from?

- ○ Entirely from the side panel.
- ○ Mostly from the side panel, occasionally from the full exam .pdf.
- ○ Mostly from the full exam .pdf, occasionally from the side panel
- ○ Entirely from the exam .pdf

Did you encounter problems with the side panel (readability, scrolling, losing view of the input form...)?

- ○ No
- ○ A few, occasionally had to adjust it
- ○ Many problems, had to constantly adjust it
- ○ It was unusable

With respect to input forms for Exercises 2 and 3, do you think it would have improved the experience using separate forms for each question, even if that would have meant showing each question in a separate section?

- ○ Yes
- ○ Probably yes
- ○ Unsure/No opinion
- ○ Probably no
- ○ No

What would you like to see improved the most?

Thanks for your feedback!

Document 1
Attached

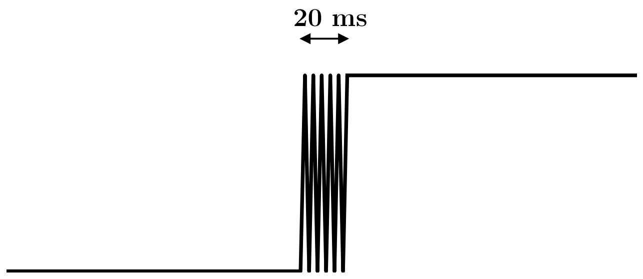
Exercise 1 Multiple choice questions (20p)

Select at most one answer for each of the following questions.

2p are given for each correct answer, 1p is subtracted for each wrong answer (the total number of points achieved in this exercise is at least 0).

1. An embedded programmer implements a switch signal listener using the function "pollSignalTask" shown bellow. The function adds RISING and FALLING events when detecting rising and falling edges of the signal, respectively. The current level of the signal is given by the macro "SIGNAL", which can be either 0 (low) or 1 (high). The function polls the signal every 10 ms. Finally, the function uses "state" to keep track of the level of the signal. **(2p)**

```
  int RISING=1, FALLING=0;
void pollSignalTask(void *params) {
  int state = 0, level;
  portTickType wakeTime;
  xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    level = SIGNAL;
    if (level == 1 && state == 0) {
      state = 1;
      xQueueSend(eventQueue, &RISING, portMAX_DELAY);
    } else if (level == 0 && state == 1) {
      state = 0;
      xQueueSend(eventQueue, &FALLING, portMAX_DELAY);
    }
    vTaskDelayUntil(&wakeTime, 10 / portTICK_RATE_MS);
  }
}
```



Assume that toggling the switch generated the above signal, which takes 20 ms to stabilize to level 1. Also assume the value of "state" was 0 before the switch toggle. What is the maximum number of events that can be generated by this signal?

1: 1 **2:** 2 **3:** 3 **4:** 4

2. Suppose a number of worker tasks, implemented by "workerTask", have access to a limited number of resources. To do the necessary work, each task consumes a single resource. The number of resources available is managed by the global variable "resourceCount". Access to resources is guarded by the semaphore "sem". **(2p)**

```
int resourceCount = 3;

void workerTask( void *params) {
  if (resourceCount > 0) {
    xSemaphoreTake(sem, portMAX_DELAY);
    resourceCount --;
    // do work consuming a single resource
    xSemaphoreGive(sem);
  }
}
```

   Assuming the initial number of resources available is 3, while the number of worker tasks is 5, what is the minimum value that "resourceCount" can take?

   1: -5 **2:** 0 **3:** -1 **4:** -2

3. How many bits can be accessed using a 1 Megabyte bit-band alias region on a 32-bit architecture? You can assume that 1 Megabyte is 2^{20} bytes, while a byte is 2^3 bits. **(2p)**

   1: 2^{10} bits **2:** 2^{15} bits **3:** 2^{18} bits **4:** 2^{16} bits

4. Suppose a structure ("struct") with l int fields, m short fields and n char fields, where $n > l + m$. What is the maximum amount of memory an element of this structure can consume on a 32-bit self-aligned architecture? Consider all possible orderings of the fields. **(2p)**

   1: $7 * l + 3 * m + n$ bytes
   2: $4 * l + 2 * m + n$ bytes
   3: $4 * (l + m + n)$ bytes
   4: None of the above

5. The smallest value that can be represented using an signed 4-bit fixed-point number with exponent $P = 2$ (i.e., numbers are represented in the form $m \cdot 2^{-2}$, where m is a signed 4-bit integer) is: **(2p)**

   1: -1 **2:** -1.75 **3:** -8 **4:** -2

6. A contract is trivially enforced if: **(2p)**

   1: its precondition is unsatisfiable
   2: its pre- and postconditions are satisfiable
   3: its precondition is valid
   4: its postcondition is valid

7. Consider the statement comprising n atomic independent conditions: **(2p)**

```
    if (b1 || b2 || ... bn)
```

Give the minimum number of times the statement should be executed in order to achieve both MC/DC and condition coverage.

1: 2 **2:** n **3:** $n+1$ **4:** $n+3$

8. Suppose a program fails on the test cases $\{1, 2, 2, 4, 5\}$, $\{1, 2, 2\}$, $\{1, 2\}$ and $\{5\}$. How many of these test cases are 1-minimal? **(2p)**

1: 1 **2:** 2 **3:** 3 **4:** 4

9. The formula $(\neg p \vee q) \wedge (q \rightarrow \neg r \wedge \neg p) \wedge (p \vee r)$ is **(2p)**

1: valid **2:** unsatisfiable **3:** satisfiable but not valid

10. Suppose a verification tool (say, a bounded model checker like CBMC) is applied to the following program: **(2p)**

```
  int ar[4];
  for (int i = 0; i < 4; ++i)
    assume(ar[i] < ar[(i+1)%4]);
  assert(ar[0] > 0);
```

Which result do you expect?

1: No assertions can fail **2:** For some inputs, the assertion fails

Question 1

Attached

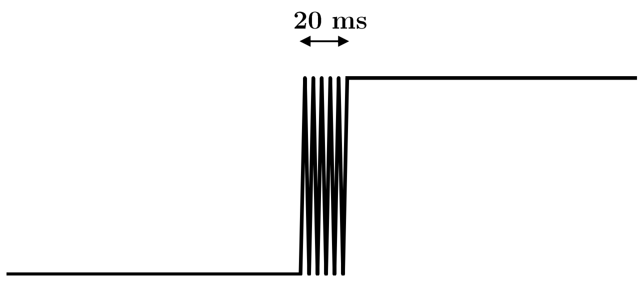
Exercise 1 Multiple choice questions (20p)

Select at most one answer for each of the following questions.

2p are given for each correct answer, 1p is subtracted for each wrong answer (the total number of points achieved in this exercise is at least 0).

1. An embedded programmer implements a switch signal listener using the function "pollSignalTask" shown bellow. The function adds RISING and FALLING events when detecting rising and falling edges of the signal, respectively. The current level of the signal is given by the macro "SIGNAL", which can be either 0 (low) or 1 (high). The function polls the signal every 10 ms. Finally, the function uses "state" to keep track of the level of the signal. **(2p)**

```
  int RISING=1, FALLING=0;
void pollSignalTask(void *params) {
  int state = 0, level;
  portTickType wakeTime;
  xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    level = SIGNAL;
    if (level == 1 && state == 0) {
      state = 1;
      xQueueSend(eventQueue, &RISING, portMAX_DELAY);
    } else if (level == 0 && state == 1) {
      state = 0;
      xQueueSend(eventQueue, &FALLING, portMAX_DELAY);
    }
    vTaskDelayUntil(&wakeTime, 10 / portTICK_RATE_MS);
  }
}
```



Assume that toggling the switch generated the above signal, which takes 20 ms to stabilize to level 1. Also assume the value of "state" was 0 before the switch toggle. What is the maximum number of events that can be generated by this signal?

1: 1 **2:** 2 **3:** 3 **4:** 4

2. Suppose a number of worker tasks, implemented by "workerTask", have access to a limited number of resources. To do the necessary work, each task consumes a single resource. The number of resources available is managed by the global variable "resourceCount". Access to resources is guarded by the semaphore "sem". **(2p)**

```
int resourceCount = 3;

void workerTask( void *params) {
  if (resourceCount > 0) {
    xSemaphoreTake(sem, portMAX_DELAY);
    resourceCount --;
    // do work consuming a single resource
    xSemaphoreGive(sem);
  }
}
```

Assuming the initial number of resources available is 3, while the number of worker tasks is 5, what is the minimum value that "resourceCount" can take?

1: -5 **2:** 0 **3:** -1 **4:** -2

3. How many bits can be accessed using a 1 Megabyte bit-band alias region on a 32-bit architecture? You can assume that 1 Megabyte is 2^{20} bytes, while a byte is 2^3 bits. **(2p)**

1: 2^{10} bits **2:** 2^{15} bits **3:** 2^{18} bits **4:** 2^{16} bits

4. Suppose a structure ("struct") with l int fields, m short fields and n char fields, where $n > l + m$. What is the maximum amount of memory an element of this structure can consume on a 32-bit self-aligned architecture? Consider all possible orderings of the fields. **(2p)**

1: $7 * l + 3 * m + n$ bytes
2: $4 * l + 2 * m + n$ bytes
3: $4 * (l + m + n)$ bytes
4: None of the above

5. The smallest value that can be represented using an signed 4-bit fixed-point number with exponent $P = 2$ (i.e., numbers are represented in the form $m \cdot 2^{-2}$, where m is a signed 4-bit integer) is: **(2p)**

1: -1 **2:** -1.75 **3:** -8 **4:** -2

6. A contract is trivially enforced if: **(2p)**

1: its precondition is unsatisfiable
2: its pre- and postconditions are satisfiable
3: its precondition is valid
4: its postcondition is valid

7. Consider the statement comprising n atomic independent conditions: **(2p)**

```
    if (b1 || b2 || ... bn)
```

Give the minimum number of times the statement should be executed in order to achieve both MC/DC and condition coverage.

1: 2 **2:** n **3:** $n + 1$ **4:** $n + 3$

8. Suppose a program fails on the test cases $\{1, 2, 2, 4, 5\}$, $\{1, 2, 2\}$, $\{1, 2\}$ and $\{5\}$. How many of these test cases are 1-minimal? **(2p)**

1: 1 **2:** 2 **3:** 3 **4:** 4

9. The formula $(\neg p \vee q) \wedge (q \rightarrow \neg r \wedge \neg p) \wedge (p \vee r)$ is **(2p)**

1: valid **2:** unsatisfiable **3:** satisfiable but not valid

10. Suppose a verification tool (say, a bounded model checker like CBMC) is applied to the following program: **(2p)**

```
  int ar[4];
  for (int i = 0; i < 4; ++i)
    assume(ar[i] < ar[(i+1)%4]);
  assert(ar[0] > 0);
```

Which result do you expect?

1: No assertions can fail **2:** For some inputs, the assertion fails

Question 2

Attached

Exercise 2 RTOS Programming and Concurrency (17p)

An Embedded Systems developer has to design the controller software for a movable bridge deployed over a river. The bridge serves two main functions: closed, it allows cars to cross the river in one direction; open, it allows ships to travel past it. The bridge stays closed as long as no ships are approaching. If an approaching ship is detected, the opening procedure is initiated once no more cars are on the bridge. At the time of detection, a barrier at the entrance of the bridge is lowered, preventing the further passage of cars. We assume lowering/raising of the barrier is instantaneous. Both the opening and the closing of the bridge take 60 seconds to complete. Once fully open, the bridge remains open for at least 120 seconds, after which it starts to close if no ships are detected the last 20 seconds. Once fully closed, the barrier is raised, making it possible for cars to cross the bridge again. The bridge can only begin opening from a fully closed state.

To achieve its functionality, the bridge is equipped with two gate sensors allowing its controller software to detect cars entering and exiting the bridge. One of the sensors is mounted on the barrier, the other is at the exit of the bridge. The sensors permit keeping track of the number of cars on the bridge at any one time. The number should be 0 for the opening operation to start. A motion sensor allows detection of incoming ships.

The bridge is controlled by GPIOA output pin 0, with 0 opening the bridge and 1 closing it. The barrier is controlled by GPIOA output pin 1, with 0 lowering the barrier and 1 raising it. The motion sensor is connected to GPIOA input pin 2, with 0 indicating no motion, and 1 indicating motion (ships) on the river. Entrance and exit gate sensors are connected to GPIOA input pins 3 and 4 respectively. An entering car is identified by a rising edge on pin 2, whereas an exiting car is identified by a falling edge on pin 3.

The system is supposed to satisfy the following safety requirements ("nothing bad can happen"):

R1: If the bridge is not closed there can be no cars on it.

R2: The bridge stays fully open for at least 120 seconds.

R3: The barrier is lowered while the bridge is not closed.

Our Embedded Systems developer has started to implement the system. Soon enough the developer realized the need for two tasks: "controlTask" for implementing the control logic; and a higher priority "carsTask" for counting cars. Moreover, the developer determined that the control system can be in one of five states: (1) CLOSED, when the barrier is raised, bridge is closed, and cars can freely cross it; (2) WAIT_EMTPY, when the barrier is lowered, bridge is closed, and the system waits for cars on it to exit; (3) OPENING, when the bridge is in the process of opening; (4) OPEN, when the bridge is fully open; (5) CLOSING, when the bridge is in the process of closing. The state of the system is stored in a global variable “status”, and the logic is built around it. The intended shape of the implementation is as follows:

```
#define PERIOD 10

// state variables
enum States {
  CLOSED=0, WAIT_EMPTY=1, OPENING=2, OPEN=3, CLOSING=4
} status = CLOSED;
int openBridge=0, raiseBarrier=1;
int isMotion=0, numCars=0;

void carsTask(void *params) {
  portTickType wakeTime = xTaskGetTickCount();
  int carIn, carOut;
  // (1) define local variables

  for (;;) {
    carIn = GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_3);
    carOut = GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_4);

    // (2) code updating numCars

    vTaskDelayUntil(&wakeTime,PERIOD/portTICK_RATE_MS);
  }
}

void controlTask(void *params) {
  portTickType wakeTime = xTaskGetTickCount();
  u32 msNoMotion=0; // counts ms while no motion
  u32 msCounter=0; // general purpose ms counter

  for(;;) {
    // read sensors
    isMotion = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_2);

    // code updating msNoMotion and msCounter
    if (!isMotion) {
      msNoMotion += PERIOD;
    } else {
      msNoMotion = 0;
    }
    msCounter += PERIOD;

    // code updating openBridge, raiseBarrier and status
    switch(status) {
      case CLOSED:
        if (isMotion) {
          status = WAIT_EMPTY;
          raiseBarrier = 0;
        }
      break;

      case WAIT_EMPTY:
    // (3) the remaining implementation
    }

    // control bridge and barrier
    GPIO_WriteBit(GPIOA, GPIO_Pin_0,
      openBridge ? Bit_SET : Bit_RESET);
    GPIO_WriteBit(GPIOA, GPIO_Pin_1,
    raiseBarrier ? Bit_SET : Bit_RESET);

    vTaskDelayUntil(&wakeTime,PERIOD/portTICK_RATE_MS);
  }
}

int main(void) {
  // some initialisation code, not shown here
  xTaskCreate(controlTask, "controlTask", 100,
              NULL, 1, NULL);
  xTaskCreate(carsTask, "carsTask", 100,
              NULL, 2, NULL);
  vTaskStartScheduler();
  return 0;
}
```

1. Complete the implementation of the two tasks according to the description above in such a way that it satisfies the requirements **R1–R3**. **You should provide the code to be inserted at locations (1), (2) and (3) in functions “controlTask” and “carsTask”. The only statements allowed are assignments (e.g. “a=a+1;”),“if/else,case,break” instructions and, in the case of (1), variable declarations. In particular, use of delay tasks (“vTaskDelay” or “vTaskDelayUntil”) will lead to no points being awarded for the respective parts. For the “carsTask”, you can assume no debouncing takes place.** **(9p)**
2. To minimize the likelihood of bugs, the developer has also decided to add a high-priority task, “safetyTask”, that checks the requirements **R1–R3** at runtime. Implement the requirements by filling location (4). You should use “assert” statements to check if conditions hold. Also, use the local variable “msOpen” to implement R2. You can ignore potential race conditions, and thus, assume that “safetyTask” always checks a consistent state of the system. **(4p)**

```
void safetyTask(void *params) {

  portTickType wakeTime=xTaskGetTickCount();
  int msOpen=0;
  for(;;) {
    // (4) code checking requirements
    vTaskDelayUntil(&wakeTime,
      PERIOD / portTICK_RATE_MS);
  }
}
```

3. The developer has noted that there are several inherent environment assumptions regarding the various sensors used. Identify two such assumptions. Note that no points are awarded for trivial assumptions such as "sensor(s) are working correctly". **(4p)**

Question 3
Attached

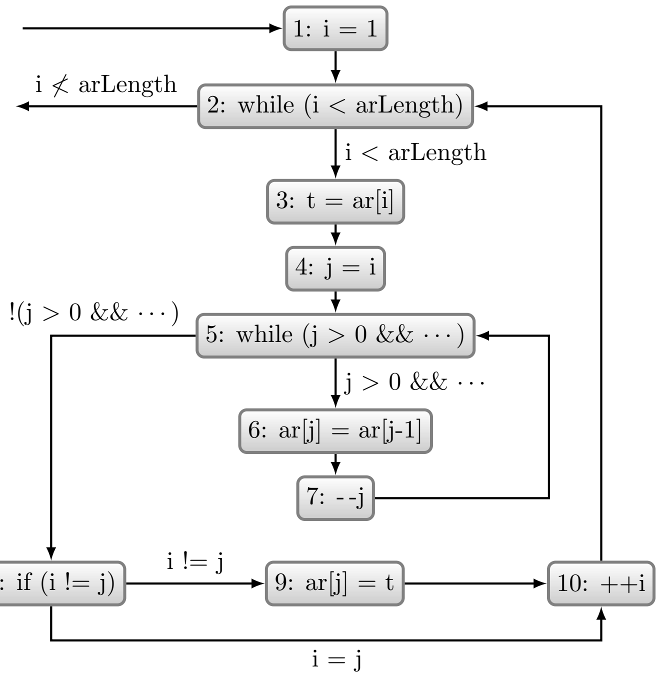
Exercise 3 Testing and Coverage (17p)

We would like to produce unit tests for the following function for sorting arrays of integers:

```
void sort(int ar[], int arLength) {
  int i, j, t;
  i = 1;
  while (i < arLength) {
    t = ar[i];
    j = i;
    while (j > 0 && ar[j-1] > t) {
      ar[j] = ar[j-1];
      --j;
    }
    if (i != j)
      ar[j] = t;
    ++i;
  }
}
```

The function receives an array of integers and the length of the array as inputs, and then swaps elements of the array until the array is sorted in ascending order.

The control-flow graph of the function is as follows:



1. We first want to derive a test oracle that checks whether the result produced by the function "sort" is correct: the function is considered to work correctly if the array "ar" is sorted in ascending order after termination of "sort". **(5p)**
   Write a corresponding test oracle by providing the body of a function

   ```
   int arrayIsSorted(int ar[], int arLength);
   ```

   that returns 1 if the given array is sorted, 0 otherwise.

2. Write a single test case that achieves full MC/DC coverage for the function "sort", considering the decisions **(6p)**

   i < arLength,   j > 0 && ar[j−1] > t,   i != j.

   The test case should be of the form

   ```
   int testCase() {
     // create inputs for sort
     // ...

     // call sort
     // ...

     return arrayIsSorted(...);
   }
   ```

   Explain in at most 6 sentences why your test case achieves MC/DC coverage.
   Note: you may assume that ar[j−1] > t is true if !(j > 0).

3. Write two additional test cases: one showing that statement coverage does not subsume (or imply) branch coverage, another showing that branch coverage does not subsume MC/DC. Justify your test cases in at most 3 sentences per test case. **(6p)**

Question 4
Attached

Exercise 4 Arithmetic (6p)

"Moving averages" are often used to smooth sensor or time series data, by computing and continuously updating the average of data read over the last N time steps. The following code gives a simple moving average implementation in fixed-point arithmetic:

```
 1 // signed 16-bit fixed-point arithmetic, P = 5,  can
 2 // represent values between -1024.0 and +1023.96875
 3 typedef s16 FIA;
 4 #define P 5
 5
 6 // standard fixed-point operations
 7 #define FADD(x, y)     ((x) + (y))
 8 #define FSUB(x, y)     ((x) - (y))
 9
10 // read data from sensor
11 FIA readData() { ... }
12 // output current average
13 void setAverage(FIA average) { ... }
14
15 // task computing the moving average
16 #define WINDOW 20
17 void averageDataTask(void *param) {
18   FIA dataHistory[WINDOW], sum = 0;
19   int pos;
20
21   // initialise history array and sum
22   for (pos = 0; pos < WINDOW; ++pos) {
23     dataHistory[pos] = readData();
24     sum = FADD(sum, dataHistory[pos]);
25     vTaskDelay(...);
26   }
27
28   for (pos = 0; ; pos = (pos + 1) % WINDOW) {
29     setAverage(sum / WINDOW);
30     vTaskDelay(...);
31     // remove old data from sum
32     sum = FSUB(sum, dataHistory[pos]);
33     dataHistory[pos] = readData();
34     // add new data to sum
35     sum = FADD(sum, dataHistory[pos]);
36   }
37 }
```

1. Analyze the use of fixed-point arithmetic in the program. Specify **all** lines in the "averageDataTask" in which rounding errors or overflows in fixed-point arithmetic can occur. Explain for which inputs rounding errors or overflows might occur, and justify that you have **(2p)**

identified all problematic statements.

2. Suppose “readData” returns the value 20.0 in the first 50 iterations, the value 100.0 in all remaining iterations. Can overflows occur during execution of the “averageDataTask”? If so identify the first iteration when an overflow happens. Motivate in at most 4 sentences that the iteration selected is indeed the first. **(4p)**
(Assume silent integer overflows, $32767 + 1 == -32768$)