name . . . . :	. . . .
first name:	. . . .
matr. no. :	. . . .

1. Question:

Evaluate the following statements and write down True or False. 10 points

true	false	
□	☑	A video game is a program that has to be designed in respect to hard real-time constraints.
□	☑	Soft real-time systems must always meet their deadlines.
☑	□	The WCRT is the longest time a task or system needs to react to a given input.
☑	□	Binary semaphores in FreeRTOS do not implement priority inheritance.
□	☑	The following stores the memory address of the variable `foo` in the variable bar, ‘`bar = *foo`‘
□	☑	The FreeRTOS scheduler ensures all tasks will meet their deadlines.
□	☑	Static memory allocation is done using the ‘`alloc`‘ family of functions.
□	☑	The tick granularity in FreeRTOS is independent of the CPU clock.
□	☑	Tasks in the Suspended state are able to be rescheduled by the scheduler.
□	☑	Every high-level programming language provides a native concept of time.

name . . . . :	. . . .
first name:	. . . .
matr. no. :	. . . .

2. Question:

For the answers on the following pages, clearly mark the question number you are answering and write a short answer for each of the following questions.

2.1 What are the four main components involved in the entire compilation process for compiling C code? 2 points

- preprocessor
- compiler
- assembler
- linker

2.2 When are `#define` directives handled in the compilation process and why are they used? 2 points

The preprocessor handles then, used to create text human readable text substitutions and defining preprocessor variables.

2.3 What part of a program is stored in the `.bss` section? 2 points

Statically allocated variables

2.4 When and why are function prototypes at the top of a C file required? 2 points

As files are processed top down, if a function is called before it has been defined then the compiler does not know how the function ”looks” and while it can take an educated guess it is best to place a function prototype at the top of the C file to let the compiler know the form of the function.

name . . . . :	. . . .
first name:	. . . .
matr. no. :	. . . .

3. Question:

Clearly mark the question number you are answering and write a short answer for each of the following questions.

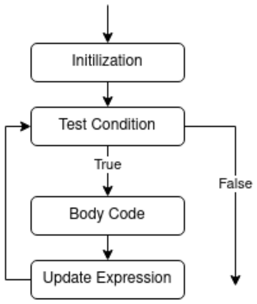
3.1 Why can pointer be used to reference arrays even if the array was defined statically using the `[ ]` notation? 2 points

Arrays are a number of sequential items in memory, each the size of the specified data type. As such a pointer to the array + some multiple of the data type size achieves the same function as the [] accessors.

3.2 How are strings implemented in C? 2 points

Null terminated array of characters

3.3 Given the following flow chat, name the type of loop that is being shown, give an example of the C code required for such a loop.



2 points

for loop

3.4 Give the return type, name and parameter types of the following C function

```c
int *copy_array_subset(int array[], char index, char length){
    ...
}

```

2 points

- Returns an integer pointer
- Is called copy_array_subset
- Takes in an integer array, a char for the index, and a char for the length

3.5 Given the following C code, fill in the missing operators (shown with _____), note that the operator might be empty, in that case place an X inside the field.

```
struct time{
  unsigned char hour;
  unsigned char minute;
  unsigned char second;
};
struct time theTime = {0};

void increment_hours(struct time *tm){
  tm_____hours++;
}

int main(void){
  ...
  increment_hours(_____theTime);
  ...
}
```

2 points

->
&

name . . . . : . . . .
first name: . . . .
matr. no. : . . . .

4. Question:

Clearly mark the question number you are answering and write a short answer for each of the following questions.

4.1 In which settings or scenarios is a real-time operating system (RTOS) required? 2 points

- Contrains from physical environment
- Predicability/concept of time
- Focus on worst-case behaviour (meet timing requirements)

4.2 Name 5 typical characteristics of embedded systems? 5 points

- Dedicated functionality
- Interaction with the physical environment
- Heterogeneous
- Constraints on power, size, manufacturing cost...
- Irregular design (hardware-software partitioning)
- No general-purpose human-machine interface

4.3 Define the WCET and the WCRT. What is the difference between the two? 3 points

- Worst Case Execution Time (WCET): Longest time a certain code needs to execute on a given platform.
- Worst Case Response Time (WCRT): Longest time, a task or system needs to react to a given input.
- WCET is part of WCRT plus other tasks, HW accesses etc.
- extra point for graph

4.4 What are Mutexes used for? Give a use case scenario, where NOT using a Mutex would lead to a problem. 4 points

- Race condition access
- Bank account example

4.5 What is priority inversion, what does it result from and how can it be solved? 2 points

Where a medium priority task inadvertently runs instead of a high priority task as a result of using a semaphore. Using a Mutex, which implements priority inheritance, will void the problem.

4.6 How can a global variable be made thread-safe. Outline the steps needed to update the value of the resource. 2 points

- A Mutex should be used to provide a locking mechanism
- Steps:
  - Mutex taken
  - Resource updated
  - Mutex returned

4.7 What is the difference between hard and soft real-time? Give an example for each.
2 points

- Hard: MUST not exceed deadline, e.g. Brake controller
- Soft: SHOULD not exceed deadline, e.g. Multimedia stream

4.8 What is the difference between preemptive scheduling and non-preemptive scheduling
2 points

- non-preemptive scheduling allows tasks to run until completed, only then will the scheduler assign the next appropriate (probably highest priority) task to the CPU
- pre-emptive scheduling allows for tasks to be interrupted during execution, allowing a higher priority task to interrupt a lower priority task and take control of the CPU

4.9 Please list the three core components a kernel
2 points

- Scheduler
- Tasks
- Inter-process communications (IPC)

4.10 What mechanism allows for multi-threaded applications to run on a single CPU core? Without going into too much detail, explain how it is achieved. 2 points

Context switching, saving the CPU's context from one task then loading another task's context and continuing execution.

name . . . . : . . . .
first name: . . . .
matr. no. : . . . .

6. Question:

6.1 Assume a direct mapped instruction cache:
Given a 32 bit system, if 12 Bits are used for the index and the cache capacity is 60Kb (1kb = 1024 Bytes). How much additional memory is needed in the cache to store the tags? Why are the tags stored? 5 points

capacity = 61440 index -> 2^{12} = 4096 cache lines -> 15 byte cache lines offset = 4 bits tag = 32 - 12 - 4 = 16 bits extra memory = 4096 * 16 = 65536b = 8192B = 8 KB

6.2 Now consider a different cache that can store 49152 Bytes of user data and has a index size of 9 bits.
a) How many cache lines are there in total?
b) How big is a single cache line?
c) How big is the Offset?
d) How big is the Tag? 6 points

- a: number of lines = 2^{index} = 512
- b: line size = cache size/256 = 49152/512 = 96
- c: offset bits = 7
- d: tag = 32 - index - block offset = 32 - 8 - 7 = 17

6.3 Using bullet points, explain the lookup process for direct mapped cache when looking for a target memory location in cache 6 points

- Go to the cache line specified by the address' index
- Compare the address' tag to that saved in the cache line's meta-data
- On hit, use the offset to access the target data
- On miss, first load the target memory block into the cache line and then use the offset to access the data

name . . . . : . . . .
first name: . . . .
matr. no. : . . . .

7. Question:

7.1 Given the following 3 tasks with their respective periods and execution times:
t1: 5, 1
t2: 4, 2
t3: 10, 3

Assuming a preemtive, rate monotonic scheduling strategy:

Use fixed point computation to determine the worst case response time of task 3

6 points

Note here I have solved using the wrong period for t1, using a 6 instead of a 5.

$\underline{T_1}$

$r_1 = 1$

$\underline{T_2}$ $r_2^0 = 2$, $r_2^1 = 2 + \lceil \frac{2}{6} \rceil 2 = 2+2=4$
$r_2^2 = 4 + \lceil \frac{4}{6} \rceil 2 = 4$

$\underline{T_3}$ $r_3^0 = 3$, $r_3^1 = 3 + \lceil \frac{3}{6} \rceil 2 + \lceil \frac{3}{4} \rceil 1 = 3+2+1 = 6$
$r_3^2 = 3 + \lceil \frac{6}{6} \rceil 2 + \lceil \frac{6}{4} \rceil 1 = 3+2+2 = 7$
$r_3^3 = 3 + \lceil \frac{7}{6} \rceil 2 + \lceil \frac{7}{4} \rceil 1 = 3+4+2 = 9$
$r_3^4 = 3 + \lceil \frac{9}{6} \rceil 2 + \lceil \frac{9}{4} \rceil 1 = 3+4+3 = 10$
$r_3^5 = 3 + \lceil \frac{10}{6} \rceil 2 + \lceil \frac{10}{4} \rceil 1 = 3+4+3 = 10$

$r_1 < t_1$, $r_2 < t_2$, $r_3 < t_3$ $\therefore$ Schedwable

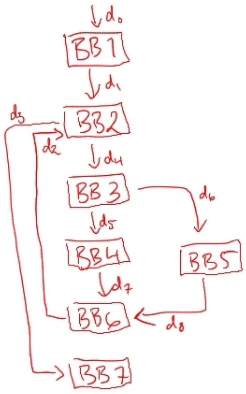
7.2 The following C code takes two sorted arrays of numbers (in1 and in2) and merges them into a new sorted array (out).
On the code snippet below, draw lines showing where the basic blocks are and label them. NOTE: Line 13 (else) does not belong to any BB.

```
 1 void merge(int in1[5], int in2[5], int out[10])
 2 {
 3    int i1 = 0;
 4    int i2 = 0;
 5    int iO = 0;
 6    while( iO < 10 )
 7    {
 8       if( i1 < 5 && (i2 >= 5 || in1[i1] < in2[i2]) )
 9       {
10          out[iO] = in1[i1];
11          i1++;
12       }
13       else
14       {
15          out[iO] = in2[i2];
16          i2++;
17       }
18       iO++;
19    }
20    return;
21 }
```

4 points

BB1: 1->5
BB2: 6
BB3: 8
BB4: 10->11
BB5: 15->16
BB6: 18
BB7: 20

7.3 Draw the control flow graph of the basic blocks found in the previous question. Explicitly mark the entry and exit point and name all edges with d0, d1, ... 7 points



7.4 How often will the basic block containing line 18 be executed? 2 points

10

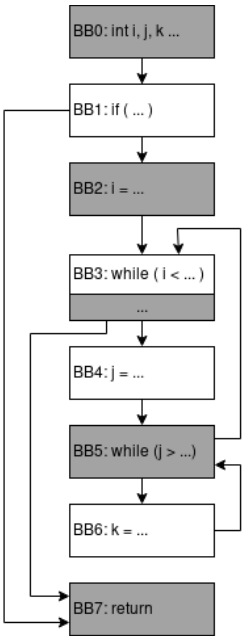
7.5 How often will the basic block containing line 15 be executed? 2 points

5

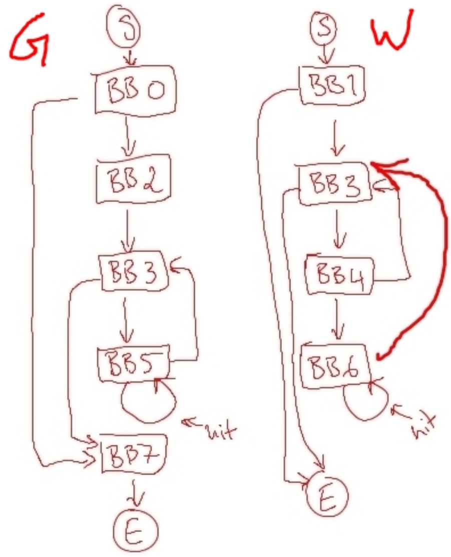
7.6 Write down the structural and logical constraints 6 points

Structural constraints
d0 = d1 = x1
d1 + d2 = d3 + d4 = x2
d4 = d5 + d6 = x3
d5 = d7 = x4
d6 = d8 = x5
d7 + d8 = d2 = x6
d3 = d9 = x7 Logical constraints
x2 = 11
x5 = 5
x6 = 10
etc

7.7 Giving the following control flow graph and basic block to cache line mappings, draw the cache conflict graphs. When do (if any) cache hits occur on the system. Note: the system has two cache lines: white and grey



8 points


G
S
BB0
BB2
BB3
BB5
hit
BB7
E
S
W
BB1
BB3
BB4
BB6
hit
E