This exam (6 pages) consists of 60 True/False questions. Your score will be computed as: \( \text{max}(0, \frac{\# \text{correct}}{60} - \frac{1}{2}) \times 2 \times 9 + 1 \)

It is **not** allowed to consult the book, handouts, or any other notes.

Instructions for filling in the answer sheet:
- You may use a **pencil** (erasures are allowed) or a **pen** (blue or black, **no** red, **no** strike outs).
- Fill in the boxes **completely**.
- Answer **all** questions; there is no penalty for guessing.
- Do not forget to fill in your **Name** and **Student Number**, and to **sign** the form.

The following abbreviations are assumed to be known:
- FQS (Function Queue Scheduling)
- ISR (Interrupt Service Routine)
- RR (Round Robin)
- RRI (Round Robin with Interrupts)
- RTOS (Real-Time Operating System)

One system clock tick = 10 ms (unless stated otherwise).

We make use of the following definitions:

```c
void delay(int ms) {
    // do some CPU computation to the number of ms milliseconds
}

void putchar(char c) {
    // while (!UART tx buffer not empty)
    ;

    // send c to UART tx buffer
}

void puts(char *s) {
    // write string s using putchar
}
```
1. Embedded programming is more difficult than “classical” programming because of the higher level of abstraction involved. true/false
2. A defining characteristic of embedded systems is the restricted, or complete lack, of a user interface. true/false
3. Despite advances in software engineering practices, as a rule of thumb, embedded software contains 1-10 bugs per million lines of code. true/false
4. An embedded program can be coded as a finite state machine where all state transitions are triggered by user actions. true/false
5. Finite State Machines can be coded in VHDL. - An advantage of doing so is that it results in lower interrupt latency as less context (e.g., registers) need to be saved and restored. true/false
6. A software interrupt is an asynchronous signal to indicate the need for a change in the execution flow. true/false
7. Besides Finite State Machines other models of computation suitable for embedded systems include Symbolic Execution and Discrete Events. true/false
8. typedef void *resolve(void *old, void *new);
   The definition above declares the prototype of the function resolve, which takes two arguments of type void * and returns a void pointer as result. true/false
9. Valgrind is programming tool that aids memory debugging. - it does so by executing a program in a safe environment. true/false
10. The C language is centered around the int data type, which is defined to hold integral numbers of at least 16 bits. true/false
11. Finite State Machines can be coded in a number of ways in C. - In the table-based solution, every transition (arc) is encoded as a separate function. true/false
12. int main(void)
    {
        int c;
        statefp state = before;
        while((c = getchar()) != EOF) {
            state = (statefp) (*state)(c);
        }
        return 0;
    }
   The above driver loop for a FSM is interrupt based. true/false
13. Specifying the type of statefp is difficult in C because forward declarations are not supported for function types. true/false
14. Using interrupts improves system response time. true/false
15. An interrupt service routine should restore the context upon exit. true/false
16. To guarantee atomicity interrupts must be disabled.  

17. An ISR can **not** be interrupted by another ISR.

18. ```c
static int iSeconds, iMinutes;
void interrupt vUpdateTime(void)
{
    ++iSeconds;
    if (iSeconds>=60) {
        iSeconds=0;
        ++iMinutes;
    }
}
long lSeconds(void)
{
    disable();
    return (iMinutes*60+iSeconds);
    enable();
}
``` 

Despite disabling interrupts the above pseudo code fails to solve the shared-data problem.

19. By structuring a program as a collection of tasks the data sharing problem is resolved.

20. An interrupt vector table contains the code of the interrupt service routines.

21. The worst-case latency for servicing an interrupt is a combination of factors, including the time taken for higher priority tasks.

22. Given the following pseudo code, which reads the current values of 3 different buttons and acts accordingly. The 3 buttons are all mapped to bits 0..2 of the button register. The buttons are already debounced.

```c
void f1(void) { delay(1000); }
void f2(void) { delay(2000); }
void f3(void) { delay(3000); }

void main (void) {
    while (1) {
        if (buttons & 0x01) f1();
        delay(1000);
        if (buttons & 0x02 ) f2();
        delay(1000);
        if (buttons & 0x04 ) f3();
    }
}
``` 

This code is an example of an RR architecture.

23. When none of the buttons have been pressed, the longest time that button #2 must be pressed to activate f2() once is 1 second.

24. When the system is in an arbitrary state, button #1 must be pressed at most 7 seconds to activate f1().

25. **Priority inversion** requires a minimum of 3 tasks of different priority and 3 semaphores to occur.  
   true/false

26. A **deadly embrace** requires a minimum of 2 tasks and 1 semaphore to occur.  
   true/false

27. Shared variables marked `volatile` guarantee atomic access.  
   true/false

28. With an RR architecture, the handling of an I/O device may need to wait until all other devices have been served.  
   true/false

29. An RRI architecture supports priority-based ISRs.  
   true/false

30. The response time to an external event in an FQS architecture depends on the longest task in the system.  
   true/false

31. Consider an alarm system that constantly monitors the digital output of several motion detector sensors in a house. If a breach is detected then an intermittent alarm sound is triggered.  
   - To guarantee a minimum response time an FQS architecture must be used.  
   true/false

32. The **primary** shortcoming of an RRI architecture is that critical sections must be used.  
   true/false

33. An FQS architecture has a smaller memory footprint than an RTOS as it needs only one stack.  
   true/false

34. In an RTOS, tasks can be in state BLOCKED, READY or RUNNING.  
   - A task can transition directly from BLOCKED to READY.  
   true/false

35. Semaphores can be used for signaling between ISRs.  
   true/false

36. A reentrant function may **not** reference variables labeled `extern`.  
   true/false

37. A semaphore used for guaranteeing mutual exclusive access to shared resources must be initialized to 1.  
   true/false

38. A high-priority task must **not** invoke an RTOS function that may block.  
   true/false

39. An ISR may call the `OS_post()` routine, provided that the RTOS “knows” that the invocation is by an ISR and not by an ordinary task.  
   true/false

40. The ‘alternating buffers’ technique addresses the shared-data problem by copying the data from the in- to the out-buffer instead of passing a pointer.  
   true/false

41. ```c
   int f (int x) {
       disable_int();
       !! read some global variables
       !! do some processing, call some functions
       !! write some global variables
       enable_int();
   }
```
   Function `f()` disables/enables interrupts to address the shared-data problem.  
   - However, when `f()` calls itself recursively, it is no longer reentrant.  
   true/false
42. Given is the following RTOS (pseudo) code with priority T1 > T2.

```c
void T1(void) {
    while (1) {
        OS_Pend(sem1); // event #1 may unblock any time
        f(1);
    }
}

void T2(void) {
    while (1) {
        OS_Pend(sem2); // event #2 may unblock any time
        f(-1);
    }
}

void f(int i) {
    delay(10); // do some computation
    counter = counter + i; // modify some global counter
    printf("%d
", counter); // print result
}
```

The function `f()` is reentrant. true/false

43. If `count` is set to 15 when event 2 occurs, and event 1 follows 3 ms later, then the first value printed is 16. true/false

44. If the call to `delay` is replaced with `OSTimeDly` the order of the print statements depends on whether or not a timer interrupt appeared in between the two events. true/false

45. An RTOS usually provides two types of delay functions: polling-based and timer-based. - timer-based delays are more efficient as other tasks can run while the caller is waiting for the specified time to pass. true/false

46. The heartbeat timer is a single hardware timer an RTOS is using to verify that the system is still progressing (i.e. not deadlocked). true/false

47. Assume that one system clock tick = 10 ms. - Calling the function `OSTimeDly(6)` causes a delay between 65 and 75 ms. true/false

48. To address the shared-data problem, many RTOSs provide communication primitives like queues, mailboxes, and pipes. - a common pitfall is that they allow pointers to be passed from one task to another. true/false

49. Even when an RTOSs is aware of which task is using which semaphore, it cannot prevent deadlock. true/false

50. Time-slicing should be avoided in an RTOS because it makes the response time of tasks less predictable. true/false

51. The minimal memory footprint of a program grows linearly with the number of tasks. true/false

52. Printing from an ISR is to be avoided except when the RTOS provides a reentrant primitive to do so. true/false
53. Time slicing between tasks of equal priority is common practice in embedded systems.  
   true/false

54. Aborting tasks is nontrivial because a task may hold resources (e.g., a semaphore) when  
   being destroyed.  
   true/false

55. Tasks should have different priorities to avoid fairness issues imposed by the RTOS.  
   true/false

56. Code coverage tools help in thorough testing.  
   - a 100% coverage implies a bug-free program.  
   true/false

57. A logic analyzer is preferred to an in-circuit emulator because it is easier to install; not all  
   signals need to be connected.  
   true/false

58. Debugging through scripting test scenarios can only be used to test HW-independent code.  
   true/false

59. A large study of outdoor sensor-network deployments [Beutel:2009] has shown that the  
   two most underestimated problems have been the water-proof packaging of the sensor  
   nodes and the provision of a reliable base station.  
   true/false

60. When debugging code for a distributed sensor network, collecting the (debug) output of  
   the nodes can be arranged in different ways.  
   - A wireless testbed requires no physical instrumentation (i.e. wiring) of the sensor node.  
   true/false