Programming for Engineers 2015 – Demo Exam

2015-xx-xx

Name: ______________________________

Matriculation Number: ______________________________

Course of Study: ______________________________ **since** ____________

Duration: **120 minutes** (2 hours)

Exam material: Writing instruments (pens only; no pencils). The supervisors can hand out extra pages if needed.

Help: Any questions? Ask the supervisors!

This exam has **7 page**. Please check that all pages are present.

In this exam you can achieve a maximum of **60 points**.
The exam is passed, if you achieve at least **30 points**.

Exercise		Max. Points	Achieved Points
1	Algorithms	12	
2	Board-Programming	20	
3	Data Structures	15	
4	Programm Analysis	8	
5	Mixed Bag	5	
Sum		60	

Points

Grade

Notes

1 Algorithms [12 Points]

The factorial function $n!$ is for natural numbers $n \in \mathbb{N}$ defined as:

$$n! = n \cdot (n-1)! = 1 \cdot 2 \cdot \dots \cdot n$$

with $0! = 1! = 1$.

Accordingly, the factorial for $n = 4$ is defined as

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$

Implement the function `fak(n)`, which computes for a given **positive integer argument** its factorial value:

```
int fak(int n) {
    // Your code
}
```

a) [4 Points] Implement `fak()` using a *loop*.

b) [4 Punkte] Implement `fak()` using *recursion*, i.e. `fak()` is calling on itself.

c) [4 Punkte] Create an *array* containing the factorial values 0! bis 5!. If `fak()` is called with input values $0 \leq n \leq 5$, you directly return the correct result from the array. Extend your previous solution .

2 Board-Programming [20 Points]

The following setup is provided to your Arduino-board

- A *bounce-free button*, attached to **pin 8** and to −(GND). If the button is pushed, the value on pin 8 is `LOW`, `HIGH` otherwise.
- A *LCD-Display*, attached to the pins 20–21, GND and VCC
- A LED, attached to Pin 13.

Extend the following code section in the subsequent exercises and implement an *electronic stopwatch*:

```
#include <Wire.h>
#include <LiquidCrystal.h>

// button
int buttonPin = 8;

// LED
int ledPin = 13;

// LCD-displaz
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);

  lcd.init();
  lcd.backlight();
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

1. If the button is pushed for the first time, the time measurement starts. The LCD shows the message „Running“.
2. If the button is pushed for the second time, the LCD shows the time between the first and the second push (in milliseconds)
3. Afterwards start over at step 1.

Use the following functions:

```
int digitalRead(int pin);                   // read Pin (LOW or HIGH)
void digitalWrite(int pin, int state);  // set pin state (LOW or HIGH)
unsigned long millis();                     // Milliseconds since program start
lcd.print(int x);                           // print number x on LCD
lcd.print(char s[]);                        // print string s on LCD
lcd.clear();                                // delete LCD
int strcmp(s, t);                           // returns 0 if s == t; otherwise !=0
```

a) [5 Points] Implement the stopwatch using the presented functions. (Hint: Read the remainder of this exercise to plan possible extensions)

b) [5 Points] Extend your stopwatch, allowing the LED to blink during a measuring process.

c) [5 Points] Extend your stopwatch in a way, that the time is shown in seconds with 3 decimal places. (Hint: First provide the integer part, followed by “.“ and the decimal places)

d) [5 Points] Extend your stopwatch in a way that it can be controlled by network access. Assume, that your Board already features an implementation of a webserver. On a requesting the url `http://Arduino-IP/PATH`, the function `void handle_request(char path[])` is called. The content of `PATH` is provided as an argument.

   The stopwatch should handle the following queries:

   – `handle_request("start")`: The stopwatch reacts as if the button was pushed the first time. The LCD monitor shows “Running“. (If the stopwatch is already running, nothing happens)

   – `handle_request("stop")`: The stopwatch reacts as if the button was pushed the second time. Time measurements stop and the LCD is showing the measured time. (Is the time measurement not running, nothing happens.

   – If `path` has another value, `path` is shown on the LCD-Display, followed by a question mark.

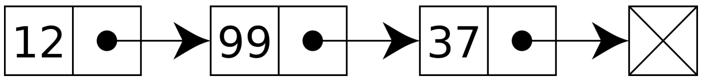
   Implement `handle_request()`. Use the introduced `strcmp()` function, to compare two strings.

3 Data Structures [15 Points]

A *linked list* is a dynamic data structure. It allows to store related objects. In contrast to arrays, the number of objects is not predefined. A list is realised by *pointers* on a subsequent element.

Dhe following figure shows a linked list containing three elements. An Element is defined as

```
struct Elem {
    int value;          // Der Wert
    struct Elem *next; // Zeiger auf das nächste Element
};
```



The pointer of the last element (i.e. 37) contains the value `NULL`.

To access a list, you start with the first element (i.e. 12) and follows the pointer to the next element. The following function checks, if the list e contains an element with value x. If this is the case, it returns a pointer to this element. If not, `NULL` is returned.

```
//First element of list with value x is returned
// (or NULL, if it is not present)
struct Elem *search(struct Elem *list, int x) {
   struct Elem *e = list; // Erstes Element
   while (e != NULL && e->value != x)
       e = e->next;
   return e;
}
```

It is your task to write a function, which appends a given element as the last element of a nonempty *list*.

a) [3 Points] Assume you want to append an element with value 44 to the list. Draw a figure, which shows the list after successfully appending item (use the same style as the sample figure).

b) [8 Points] In order to append an element, you first need to find the last element. Develop the function `last()`, which (similar to the introduced `search()` functin) iterates through the list and returns the pointer of the last element.

```
// Returns last element of a given list
struct Elem *last(struct Elem *list) {
    // Your Code
}
```

c) [4 Points] You are given an existing element E. Use your function `last()`, to append E at the end of the list. Implement the function`append()` accordingly. Be careful: the pointer of the last (new) element must be `NULL`.

```
// append element E to list LIST
void append(struct Elem *list, struct Elem *e) {
    // Your Code
}
```

4 Program Analysis [8 Points]

Some software bugs are obvious, some tend to be more subtle. Each of the following functions is supposed to return the value 42, but contains a bug. Find all the bugs in each function and ...

a) ... describe what the function is actually doing and what its problem is? (“Syntax-Errorr“, “Returns 41“, etc.)

b) ... provide a minimal solution to solve the bug

```
int f1() {
    return 43;
}

int f2() {
    return 42
}

int f3() {
    if (5 < 2);
        return 43;
    return 42;
}

int f4() {
    int n = 0;
    while (n <= 42)
        n++;
    return n;
}

int f5() {
    int a[] = {40, 1, 1};
    int sum = 0;
    for (int i = 1; i < 3; i++)
        sum = sum + a[i];
    return sum;
}

int f6() {
    int n = 21;
    for (int n = 0; n < 1; n++)
       n = n * 2;
    return n;
}

int f7() {
    int x = 0;
    while (x < 1000) {
        if (x == 42)
            return x;
    }
    return 42;
}

int f8() {
    int n = 42;
    if (n = 43)
        n = 44;
    return n;
}
```

5 Mixed Bag [5 Points]

Validate the following statements. Cross “T“, if the statement is correct, “F“, if it is wrong.
Bewertung:

- $+\frac{1}{2}$ Points for a correct answer
- ± 0 Points, no answer
- $-\frac{1}{2}$ Points for an incorrect answer.

1. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ An *algorithmus* consists of a sequence of steps
2. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ An `int`-variable may contain arbirtrary large values
3. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ A *loop* can execute statements repeatedly
4. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ The *von-Neumann-Architektur* seperates program- and data memory.
5. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ A non-constant *global variable* may be changed by any function
6. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ If a C-programm addresses a field outside its boundaries, it terminates immediately
7. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ An *IP-address* makes a device reachable in a computer network
8. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ The *HTTP*-Protocol describes the document structure of web pages
9. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ A function with return type `void` always returns `NULL`
10. $\boxed{\textbf{T}}$ $\boxed{\textbf{F}}$ A (non-empty) *library* typically provides additional functionality.