31 Januari 2014, 9:00-12:00h

- Write on each sheet of paper your name, student number and study (discipline). Number each sheet, and write on the first sheet the total number of sheets.
- The exam consists of 5 problems. The problems 1, 3, 4 and 5 are worth 20 points each, problem 2 is worth 10 points. You will get 10 points for free (total 100 points).
- You are not allowed to consult any literature. You are allowed to use a (graphical) calculator.
- Write neatly and carefully with a pen (do not use a pencil). If the handwriting is unreadable, or needs guessing to make something out of it, then the answer is rejected.
- Success!

Problem 1: Assignments (20 points)
For each of the following annotations determine which choice fits on the empty line (.....). The variables `x`, `y` and `z` are of type `int`. Note that `X`, `Y` and `Z` (uppercase!) are specification-constants (so not program variables).

```
1.1  /* x == 2*X */
     .....
     /* x == 42*X + 21 */

    (a) x = x/21 - 21;
    (b) x = 21*x + 21;
    (c) x = (x-21)/21;
```

```
1.2 /* x == 42*X + 21 */
    .....
    /* x == 2*X */

    (a) x = x/21 - 21;
    (b) x = 21*x + 21;
    (c) x = (x - 21)/21;
```

```
1.3 /* x == X*X*X, y == X*X, z == X */
    .....
    /* x == (X+1)*(X+1)*(X+1) */

    (a) x = 3*x + 3*y + z + 1;
    (b) x = x + 3*y + 3*z + 1;
    (c) x = x + y + 3*z + 3;
```

```
1.4 /* x == X+Y, y == 2*X-7 */
    x = 2*x - y; y = (x - 7)/2;
    .....

    (a) /* y == Y */
    (b) /* y == X */
    (c) /* x == X */
```

```
1.5 /* x == X, y == Y */
    x = x + y; y = 2*(x - y); x = (2*x -y)/2;
    .....

    (a) /* x == 2*X, y == Y */
    (b) /* x == 2*Y, y == 2*X */
    (c) /* x == Y, y == 2*X */
```

```
1.6 /* y == X, z == X + Y, x == X + Y + Z */
    z = z - y; y = x - y; x = x - z;
    .....

    (a) /* x == X+Z, y == Y+Z, z == Y */
    (b) /* x == X+Y, y == X+Z, z == Z */
    (c) /* x == Y+Z, y == X+Y, z == X */
```

Problem 2: Find the 5 errors (10 points)

The following program fragment can be used to factorize integers. There are, however, five errors in this implementation. Find them, and give of each error the line number and a correction.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int readNumber() {
 5   int input;
 6   do {
 7     printf("Please, type a natural number >1 (1=stop): ");
 8     scanf("%d", &input);
 9   } while (input < 1);
10 }
11
12 int intLog(int n, int base) { /* returns greatest e such that n%(base^e) == 0 */
13   int e=0;
14   while (n % base == 0) {
15     n /= base;
16     e++;
17   }
18   return e;
19 }
20
21 int power(int number, int exponent) { /* returns number^exponent */
22   while (exponent != 1) {
23     if (exponent%2 == 0) {
24       number = number*number;
25       exponent = exponent/2;
26     } else {
27       m = m*number;
28       exponent--;
29     }
30   }
31   return m;
32 }
33
34 void factoriseNumber(int number) {
35   int divisor = 1, first = 1;
36   while (number != 1) {
37     if (number % divisor == 0) {
38       int exponent = intLog(number, divisor);
39       if (first == 0) {
40         printf(" * ");
41       }
42       first = 0;
43       printf ("%d^%d", divisor, exponent);
44       number /= power(divisor, exponent);
45     }
46     divisor++;
47   }
48   printf("\n");
49 }
50
51 int main(int argc, char *argv[]) {
52   number = readNumber();
53   while (number > 1) {
54     factoriseNumber(number);
55     number = readNumber();
56   }
57   return 0;
58 }
```

Problem 3: Time complexity (20 points)
In this exercise the specification constant `N` is a non-zero natural number (i.e. `N>0`). Determine for each of the following program fragments the sharpest upper limit for the number of calculation steps that the fragment performs in terms of `N`. For an algorithm that needs `N` steps, the correct answer is therefore $O(N)$ and not $O(N^2)$ as $O(N)$ is the sharpest upper limit.

1.
```
int i = 0, j = 0;
while (i < N) {
  j++;
  i += 2*j - 1;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2.
```
int i=N;
while (i > 1) {
  if (i % 2 == 0) {
    i /= 2;
  } else {
    i++;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.
```
int i = 0, j = N;
while (i < j) {
  i++;
  j--;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int i, j, sum = 0, prod = 1;
for (i = 0; i < N; i++) {
  sum++;
}
for (j = 1; j < N; j++) {
  prod = prod * j;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

5.
```
int i, j, sum = 0;
for (i = 0; i < N; i++) {
  for (j = 1; j < N; j++) {
    sum++;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6.
```
int i, j, sum = 0, prod = 1;
for (i = 1; i < N; i++) {
  prod *= i;
}
j = 0;
while (prod > 0) {
  prod = prod/2;
  j++;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Problem 4: Iterative algorithms (10+10 points)
(a) The integer 145 is a curious number, as $1! + 4! + 5! = 1 + (1 \times 2 \times 3 \times 4) + (1 \times 2 \times 3 \times 4 \times 5) = 1 + 24 + 120 = 145$. Write a function `isFactorialSum(int n)` that determines whether `n` is equal to the sum of the factorial of its digits. Note that `9!=362880`, so a number with `k` digits can at most attain the sum `k*362880`. As a consequence, if `n >= 10000000` (8 digits) the result of `isFactorialSum(n)` will surely be `0`.

(b) The famous mathematician Christian Goldbach posed the following conjecture:

Every odd non-prime integer greater than 1 can be written as the sum of a prime number and twice a square.

Here are a few examples: $9 = 7 + 2 \times 1^2$, $15 = 7 + 2 \times 2^2$, $21 = 3 + 2 \times 3^2$, $25 = 7 + 2 \times 3^2$. The conjecture turned out to be false. Write a program fragment that finds the smallest odd non-prime integer `N` (where `N >1`) that cannot be written as the sum of a prime and twice a square. The time complexity of your solution should not exceed `O(N*N)`.

Problem 5: Recursive algorithms (5+15 points)
(a) Write a recursive function `add` with the following prototype: `int add(int a, int b);`
The function call `add(a, b)` should return the value `a+b`. You are not allowed to use loops (only recursion), and you are only allowed to use the operations 'add one' and 'subtract one'. You may assume that `b` is a non-negative integer, but `a` might be negative.

(b) Consider some (unsorted) integer array `a[]`. Suppose that we want to find the $\mathtt{k}^{\text{th}}$ smallest element of this array, in other words the element at index `k` after sorting the array in ascending order. Note that the $\mathtt{0}^{\text{th}}$ smallest element is the minimum of the array. For example, if the array is {`3, 6, 2, 7, 0, 1, 7, 9, 6, 8, 0`} and `k=5`, the $\mathtt{5}^{\text{th}}$ smallest element would be 6.

Sorting the entire array, however, is not necessary. We can find the element using the following idea. We choose the first element of the array as a pivot, which in this example is 3. Next, we reorder the elements of the array as follows. All elements less than 3 are placed to the left of the value 3, while all elements greater or equal than 3 are placed to the right of 3. For this example, a valid reordered array would be {`2, 0, 1, 0, 3, 6, 7, 7, 9, 6, 8`}, where the value 3 is placed at index 4. Since we search the $\mathtt{5}^{\text{th}}$ smallest element, we know that we need to continue our search to the right of index 4. In fact, we now know that we need to search for the $\mathtt{0}^{\text{th}}$ smallest element in the subarray containing the values {`6, 7, 7, 9, 6, 8`}. This idea yields a recursive approach to solving the problem. Work out the details yourself.

Write a recursive function `int kthSmallest(int k, int len, int a[])` that determines the `k`th smallest element in the array `a` with length `len`. You may assume that `k < len`.

[Tip: write a (non-recursive) function `partition` that reorders the element of the array and that returns the index of the pivot. Use this function in your recursive algorithm. If you do not succeed in writing the function `partition`, you may assume that it exists, and write only the recursive algorithm (an get half of the points, provided it is correct)]