Technical University of Denmark

Written examination, December 7, 2016	
Course name:	Mathematical software programming
Course number:	02635
Aids allowed:	All aids allowed
Exam duration:	4 hours
Weighting:	80/100

Final exam
Mathematical Software Programming

This exam contains a total of 20 questions: 16 multiple choice questions (questions 1–16) and 4 programming questions (questions 17–20). Your exam answers must be submitted electronically as a **PDF document**. You may include your code in the document along with your answers or submit the code separately in a ZIP file.

1. (2 points) The associative property

$$(a + b) + c = a + (b + c)$$

always holds for

(a) floating point arithmetic with floating point numbers a, b, and c (e.g. `double`)

A. True

B. False

(b) integer arithmetic with integers a, b, and c (e.g. `int`)

A. True

B. False

2. (2 points) Consider the following system of equations with two unknowns x and y

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

where a, b, c, d, e, and f are real numbers. Recall that if the system matrix is nonsingular (i.e., the determinant is nonzero), the solution can be expressed as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{ad - cb} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix}.$$

Catastrophic cancellation is likely to occur in the above computations if

A. $|ad - cb|$ is large

B. $|de - bf|$ or $|af - ce|$ is large

C. $a = d = 1$ and $c = b = 10^{-8}$

D. $ad \approx cb$ and/or $de \approx bf$ and/or $ce \approx af$

3. (4 points) Consider the following nested loops

```
for(i=0; i<M; i++)        // outer loop
    for(j=0; j<N; j++)    // inner loop
        a[i] = j*a[i];
```

where `a` is an array of length `M`.

(a) References to `a` are temporally local with respect to the outer loop.

A. True

B. False

(b) References to `a` are spatially local with respect to the outer loop.

A. True

B. False

4. (2 points) Cache memory is generally slower than main memory (RAM).

A. True

B. False

5. (2 points) When parallelizing a program, the speedup of the execution time using p cores is defined as

$$S(p) = \frac{T(1)}{T(p)}$$

where $T(p)$ denotes the execution time with p cores. What is the maximum speedup that is possible using any number of cores if only 80% of the program can be parallelized?

A. 1.25

B. 1.6

C. 4

D. 5

6. (2 points) Memory leaks are possible in C but not in C++.

A. True

B. False

7. (6 points) Consider the recursive function definition

$$f(0) = 0, \quad f(n) = f(n-1) + 2n - 1, \; n \geq 1,$$

where n is a positive integer. This function can be implemented in C as follows:

```
unsigned long fun(unsigned long n) {
   if (n == 0) return 0;
   return fun(n-1) + 2*n - 1;
}
```

(a) What type of recursion is this?

A. Single recursion

B. Double recursion

C. Multiple recursion

(b) What is the time complexity of this implementation?

A. $O(1)$

B. $O(n)$

C. $O(n^2)$

D. $O(2^n)$

(c) What is the space complexity of this implementation?

A. $O(1)$

B. $O(n)$

C. $O(n^2)$

D. $O(2^n)$

8. (2 points) Templates in C++ facilitate

A. functional programming

B. object-oriented programming

C. generic programming

D. specific programming

9. (2 points) Consider the following C function that evaluates the function

$$f(x) = ax^2 + bx + c$$

```
double * quad_eval(double a, double b, int double c, double x) {
    double *pv;
    double val = b;
    pv = &val;
    val += a*x;
    val *= x;
    val += c;
    return pv;
}
```

There is a problem with this implementation. What is the problem?

A. The result is subject to catastrophic cancellation when $|a|$ is large

B. The function returns a pointer to a local variable.

C. The function returns b instead of $f(x)$

D. The function leaks memory

10. (2 points) Dynamic memory allocation is faster than automatic memory allocation.

A. True

B. False

11. (2 points) The operator :: in C++ (e.g. `std::cout`) is referred to as the

A. member access operator

B. scope resolution operator

C. namespace operator

D. function operator

12. (2 points) Consider the following C program:

```
#include <stdlib.h>
#include <stdio.h>
#define N 50

double * init_array(size_t n, double h) {
   double * p = malloc(n*sizeof(*p));
   if (p != NULL) {
      for (size_t i=0;i<n;i++)
         *(p++) = i*h;
   }
   return p;
}

int main(void) {

   double *p = init_array(N);
   if (p == NULL) return EXIT_FAILURE;

   for (size_t k=0;k<N;k++) printf("%3d %6g\n",k,p[k]);

   free(p);
   return 0;
}
```

Is there a problem with this program? If yes, what is it?

A. Yes, `init_array` returns a local variable.

B. Yes, the return value of `init_array` does not point to the first element of the array.

C. Yes, `init_array` should call `free` before returning to prevent a memory leak.

D. No, the program is fine.

13. (2 points) When could it be necessary to know the endianess of a computer?

A. When sharing binary data or files between different computers.

B. When reading input from the user (e.g. with `scanf`).

C. When reading or writing numbers from/to a text file.

D. When allocating memory for multi-byte data types.

14. (6 points) Recall that a floating point number x can be represented as

$$x = s \cdot (d_0.d_1d_2 \ldots d_n)_b \cdot b^e = s \cdot \sum_{i=0}^{n} d_i b^{e-i}$$

where s denotes the sign, e is the exponent, $b \geq 2$ is the base, $d_0, \ldots, d_n$ comprise the mantissa and $d_i \in \{0, 1, \ldots, b-1\}$.

(a) What characterizes a *normal* floating point number?

A. $b = 1$
B. $e = 0$
C. $s = +1$
D. $d_0 \neq 0$

(b) What is the smallest representable number greater than 1.0?

A. $1 + b^{-(n-1)}$
B. $1 + b^{-n}$
C. $1 + b^{-(n+1)}$
D. $1 + b^{-(n+1)}/2$

(c) What is the largest representable number less than 1.0?

A. $1 - b^{-(n-1)}$
B. $1 - b^{-n}$
C. $1 - b^{-(n+1)}$
D. $1 - b^{-(n+1)}/2$

15. (2 points) Consider the following piece of code:

```
double data[4] = {5.0,-2.0,2.0,0.0};
data[1] *= 2.0;
--data[1];
```

The first element of the array is initialized as 5.0. What is the value of this element of the array `data` after running the code?

A. -5.0
B. 5.0
C. 9.0
D. 10.0

16. (2 points) Consider the following C struct and array:

```
typedef struct matrix {
    size_t m;
    size_t n;
    double **A;
} matrix_t;

matrix_t mat_arr[10];
```

Which of the following options is a correct way of assigning the value 4 to the field `m` of the second element of the array `mat_arr`?

A. `(mat_arr+1).m = 4;`

B. `*mat_arr[1].m = 4;`

C. `mat_arr[1]->m = 4;`

D. `(mat_arr+1)->m = 4;`

E. `*(mat_arr+1)->m = 4;`

17. (12 points) The cumulative sum of the elements of a vector $x = (x_1, \ldots, x_n)$ can be expressed as

$$y_i = \sum_{k=1}^{i} x_k, \quad i = 1, \ldots, n.$$

(a) Implement a function that computes the cumulative sum. Your function should have the following prototype:

```
void cumsum(
    const double * x,    /* pointer to x[0]    */
    double * y,          /* pointer to y[0]    */
    unsigned long n      /* length of arrays   */
);
```

The inputs `x` and `y` should be pointers to the first element of two non-overlapping arrays (corresponding to x and y), and the third input `n` represents the length of the arrays.

(b) What is the time complexity of computing the cumulative sum?

(c) Explain how you tested your function and/or include your test program(s).

18. (4 points) Recall that a sparse matrix can be stored in triple form which can be represented using the following abstract data type:

```
typedef struct sparse_triplet {
   unsigned long m;   /* number of rows                  */
   unsigned long n;   /* number of columns               */
   unsigned long nnz; /* number of nonzeros              */
   unsigned long * I; /* pointer to array with row indices    */
   unsigned long * J; /* pointer to array with column indices */
   double * V;        /* pointer to array with values         */
} sparse_triplet_t;
```

Write a function that transposes such a sparse matrix in-place (i.e., without allocating a new `sparse_triplet_t`). Your function should have the following prototype:

```
void trans_sparse_triplet(sparse_triplet_t * sp);
```

19. (12 points) An exponential moving average of a series of numbers $x_0, x_1, x_2, \ldots$ is defined recursively as

$$y_0 = x_0, \quad y_k = \alpha x_k + (1-\alpha) y_{k-1}, \; k \geq 1$$

where $\alpha \in (0,1)$ is a constant.

(a) Write a function that computes an exponential moving average of a series of numbers, $x_0, x_1, \ldots, x_{n-1}$, stored as an array of length n. Use the following prototype:

```
void ema(
   const double * x,     /* pointer to x[0]     */
   double * y,           /* pointer to y[0]     */
   unsigned long n,      /* length of arrays    */
   double alpha          /* constant alpha      */
);
```

(b) What is the time complexity of computing an exponential moving average?

(c) Explain how you tested your function and/or include your test program(s).

20. (10 points) Two vectors x and y of length n are orthogonal if and only if their inner product

$$x^T y = \sum_{i=1}^{n} x_i y_i$$

is equal to zero.

(a) Write a function that checks if two vector are orthogonal. Use the following prototype:

```
int orthogonal(
   const double * x,     /* pointer to x[0]     */
   const double * y,     /* pointer to y[0]     */
   unsigned long n       /* length of arrays    */
);
```

The inputs x and y should be pointers to the first element of two arrays of length n. The output should be equal to 1 if the vectors are orthogonal, and otherwise the output should be 0.

(b) Test your function with the following vectors as input:

$$x = \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix}, \quad y = \begin{pmatrix} 3 \\ -1 \end{pmatrix}.$$

Does your function work as expected? Why/why not?