Last name, first name, student ID:

Advanced Programming–WT12/13

General Instructions

Material:

You may use only printed and hand-written helping material.

No electronic devices are allowed.

Use only the exam paper that was handed out to solve the exercises. In case the space on a page is not enough, mark that you continue with your solution and use the reverse side of the preceding page.

If you require sheets for additional notes and sketches that are not to be handed in, you can obtain additional sheets from the lecturers. Please erase drafts if they shall not be taken into account.

Do not use *pencil, or red or green ink*!

Errors and ambiguities:

If you think that a question contains an error or ambiguity, then correct it or choose an interpretation that allows you to complete the exercise.

Make sure to point out your correction or disambiguation in your solution!

No questions, esp. concerning errors or ambiguities, will be answered during the exam.

General hint:

Read through all questions first. Start with the easy ones.

Often, sub questions can be solved without the results from the previous questions; if you get stuck with one problem, skip it and continue immediately with the next one.

Take the number of credits of a question into account before you start to answer. Few credits require short sentences.

Upper limits on number of bullet points/sentences/words are mandatory.

Working time:

90 minutes

Please switch off your cell phones!

1	2	3	4	5	6	7	$\sum$	Grade
/3	/3	/6	/7	/6	/8	/7	/40	

Last name, first name, student ID:

1 Assigning values (3 credits)

Write down the values of the variables *a*, *b*, *c*, *d*, and *e* at the end of each line. In case that the value of a variable is unknown and not defined by C++, use $\perp$. The lines are executed one after another, i.e. values from line ten determine the result in line eleven.

	a	b	c	d	e	f
`a=1; b=2; c=3;`						
`b=a!=1;`						
`c=b++;`						
`d=(--c-1)+2;`						
`e=(a&1);`						
`f=(e%1);`						

2 Exclusive or (3 credits)

Besides the logical binary operators *or* and *and*, there is also an operator *xor* (exclusive or). It is denoted as ^ and its (strict) table reads as follows:

a	b	a xor b
false	false	false
true	false	true
false	true	true
true	true	false

1. What is the difference of a strict *and* and a non-strict *and*? (Explain with one sentence each)

2. How do you distinguish a strict from a non-strict *or* syntactically?

3. Is there a non-strict *xor*? Explain with one sentence.

3 Flynn's Taxonomy (6 credits)

We study a processors with either a single core or dual cores with core A and a core B. Each core has four registers. The table columns below display the content of the registers in each core. All registers' initial value shall be 0 (compare first line of table). Each line then represents the content of the registers r0 to r3 after an instruction of a microprocessor has terminated, i. e. the table illustrates a register state transitions. We assume that all the instructions of each particular system belong only to a single class of Flynn's taxonomy. While Flynn's taxonomy classifies hardware, it also implies which type of operations are available. All questions below refer to these operations.

3.1 System A

Core A				
instr./reg.	r0	r1	r2	r3
	0	0	0	0
0	3	3	3	3
1	1	2	3	4
2	3	6	9	12
3	6	9	12	15

1. To which group of Flynn's taxonomy do these instructions belong to?

2. Give a short explanation (less than 20 words).

3. Give two examples for such an architecture, instruction set, or architectural extension in real world.

3.2 System B

Core A				
instr./reg.	r0	r1	r2	r3
	0	0	0	0
0	0	0	0	0
1	1	0	0	0
2	1	2	0	0
3	3	3	3	0

Core B				
instr./reg.	r0	r1	r2	r3
	0	0	0	0
0	1	0	0	0
1	1	2	0	0
2	1	2	3	0
3	4	4	4	4

1. To which group of Flynn's taxonomy do the instructions of one core belong to?

2. To which machine type (according to Flynn) does the whole dual core architecture belong to?

3. What means SPMD?

4 Everybody ~~hates~~ loves debugging (7 credits)

```
#include <iostream>
int main(int argc, char *argv[])
{
        unsigned int c;

        int a[10];

        // setup array content
        for (c=1; c<11; c++)
                a[c-1] = (int)c;


        // output reversed array content
        int *b = &a[9];  // use pointer on last array element
        for (c = 9; c >= 0; c--) {
                std::cout << *b << " ";
                b--;
        }
        std::cout << std::endl

        return 0;
}
```

Given is the C source code above. The description of the code's semantics/specification reads as follows:

- The array a has to be filled with numbers from 0 to 9.
- The content of the array should be printed to the console in reversed way.

Furthermore, the programmer may not use any array access operator for a such as the brackets [] in the for loop printing the array. Instead, the pointer b has to be used.

1. Describe one possible terminal output of the application (first eleven elements).

2. Why might the second for loop never terminate?

3. Provide one possible bugfix for the code. Add it directly to the source code above.

4. Of what kind of type is this bug? (two words)

5 Call-by-value vs. call-by-reference (6 credits)

Subject of study is the (valid) C++ code below.

```
#include <iostream>

class A {
  public:
    int i;

    A() {
      i = 0;
    }
};

void fun(A a)
{
        a.i += 1;
        std::cout << a.i << std::endl;
}

int main(int argc, char *argv[])
{
        A a;
        fun(a);    // function call
        std::cout << a.i << std::endl;
        return 0;
}
```

1. Write down the terminal output of the program.

2. What happens on the call stack in the line commented with $//function\ call$.

3. Rewrite function fun and the code in its scope to use call-by-reference instead of call-by-value. The output has to remain the same, and the application's overall semantics (states) have to be preserved as well.

6 Towers of Hanoi (8 credits)

Towers of Hanoi is a mathematical game with several existing solutions. It consists of three towers and some disks of different sizes. The goal of the game is to move all disks from tower 1 to tower 3. At the end, all disks should be positioned in decreasing order. Only one disk is allowed to be moved at a time and in the process of moving the disks, one is not allowed to place a larger disk on top of a smaller one. In the present assignment, the exact game mechanics are not relevant. However, we examine the formula to calculate the minimum moves needed for x disks:

$$hanoi(x) = \begin{cases} 1, & \text{if } x = 1, \text{or} \\ 2 \times hanoi(x-1) + 1, & \text{if } x > 1. \end{cases} \quad (1)$$

1. Write a recursive method $hanoi$ returning the value for any given $x > 0$. Input values smaller than 0 don't have to be considered.

Last name, first name, student ID:

2. Add a static variable k to the recursive function, initialize it with zero and increment it each time the function is called. (add a comment in the your source code and write code that should replace this comment here)

3. What is the value of k for general input, i.e. what is its semantics?

4. What would be the value of k if it were not declared as static? Why?

7 Lazy allocation (7 credits)

Large data-objects can prohibit programs from being executed. One possible way to circumvent this problem is to use lazy allocation. Here, the memory is only allocated if it is required not before. Subject of study is a linear algebra class that implements such a lazy allocation. It realizes matrix storage and matrix operations.

```
#include <iostream>

template <int N, int M>
class CMatrix
{
  float *data;

  CMatrix() :

    /*a*/------------------
  {

    /*a*/------------------
  }

  virtual ~CMatrix()
  {

    /*a*/------------------
  }


  /*
   * This method copies all data stored in matrix i_m
   * to the local class.
   */
  void copyFromOtherMatrix(CMatrix<N,M> &i_m)
  {
    setup();

    for (size_t i = 0; i < N*M; i++)
      data[i] = i_m.data[i];
  }
```

```
  /*
   * This method should be executed when the matrix data
   * is not required anymore.
   */
  void free()
  {

    /*a*/_________________

    /*a*/_________________
  }

  void setup() {

    /*a*/_________________

    /*a*/_________________

    /*a*/_________________

    /*a*/_________________
  }
};
```

1. Extend the given source code at the lines marked with `/*a*/` to allocate the data at appropriate points. Some lines with `/*a*/` might remain empty. Consider a memory-leak free implementation! The memory to store a matrix may only be allocated when it is really used. Visibility markers are and can be omitted.

Last name, first name, student ID:

2. May we make the function $copyFromOtherMatrix$ const? Explain your decision with one sentence.

3. $copyFromOtherMatrix$ basically realizes a copy constructor. Rewrite it as copy constructor.

4. $copyFromOtherMatrix$ realizes call-by-reference. Write down the operation's signature (only the one line) with call-by-const-reference.