

Informatik für Mathematiker und Physiker HS14

Exercise Sheet 13

Submission deadline: 15:15 - Tuesday 16th December, 2014
Course URL: <http://lec.inf.ethz.ch/ifmp/2014/>

Exercise 1 (4 points)

You are given the base class

```
struct statement {  
    virtual void execute () {};  
};
```

to represent statements in a computer program. Derive at least the following classes from this base class:

```
// variable definition with initial value  
struct variable;  
  
// increment a variable by the value of another variable, or by one  
struct increment_variable;  
  
// print the value of a variable  
struct print_variable;  
  
// sequence of commands  
struct block;  
  
// repeat a command n times  
struct repeat;
```

Use these classes to write a meaningful program of your choice. Here is an example (*Kleiner Gauss*) that also shows how the above classes are meant to be used.

```
// make statements  
variable s = 0;  
variable i = 1;  
increment_variable next_s (&s, &i);  
increment_variable next_i (&i);  
block update;  
update.add (&next_s);  
update.add (&next_i);  
repeat kleiner_gauss (&update, 100);  
print_variable result (&s);  
  
// execute statements  
kleiner_gauss.execute();  
result.execute();
```

Challenge (8 points)

Derive a class `ifelse` from the base class `statement` given in the previous exercise. For this statement you may compare two objects of type `statement`, and the operator may be given as a `char`.

```
// execute one of two statements based on the result of a
// Boolean comparison
struct ifelse;
```

Exam-Style Questions

This exercise consists of old exam questions, rewritten to give you an idea of how the computer-based exam questions might look like. In square-brackets [...] a reference to the initial exam question is given. Since the solutions of these questions are already on the website your exercise class teacher is supposed to correct your submissions only if you attest that you didn't look the solutions up.

a) [Winter 2014, Ex.1]

For each of the following 5 expressions, write C++ type and value in the corresponding gap! Assume that `x` has type `int` and value 3 *at the beginning of each subtask*.

Expression	Type	Value
<code>0 < 1 && 1 != 1 ++x > 3</code>		
<code>12.0 / 3u / 2u</code>		
<code>2 + x++</code>		
<code>2014 % 4 + x % 2</code>		
<code>x / 2 + 9.0f / 6</code>		

b) [Winter 2014, Ex.2]

For each of the following three code fragments, write down the sequence of numbers that it outputs!

a)

```
for (int i=0; i<100; i*=2)
    std::cout << ++i << " ";
```

 Output: _____

b)

```
int i=0;
int j=5;
while (i != j)
    std::cout << (i+=2) - j++ << " ";
```

 Output: _____

```

c) void f(unsigned int x) {
    if (x == 0)
        std::cout << "-";
    else {
        std::cout << "*";
        f(x-1);
        std::cout << "*";
    }
}
f(5);

```

Output: _____

c) [Winter 2014, Ex.4]

Your task is to provide an implementation of a function that computes a floating-point approximation of the exponential function

$$e^x := \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

for a given real number x . The implementation should be based on this definition of e^x as an infinite series. To obtain an approximation of e^x , the function should sum up initial terms of this series (in double precision floating-point arithmetic) *until* adding the current term does not change the value obtained so far anymore. You are *not* allowed to use any external libraries. Per gap you are only allowed to write one programming line¹ (you may also leave gaps empty).

```

double exp (double x) {
    double t = 1.0;    // 1/i!
    double ex = 1.0;  // i-th approximation of e
    double xi = 1.0;  // x^i
    double ex_old = 0.0;
    ...
    ...
    ...
    ...
    ...
    ...
    ...
}

```

¹By a programming line we mean here (i) *one* statement or (ii) the statements and expressions needed for *one* compound statement as e.g. the head-line of a for-loop.

d) [Summer 2011, Ex.2]

The following questions shall be answered with either *yes* or *no*. In order to obtain points for this exercise you have to achieve at least 50% of the maximum number of points of this exercise. But for wrong answers you will *not* lose points.

- a) There is no difference for the user between the following two declarations. Yes, No

```
class foo {                struct foo {
    int i;                  int i;
};                          };
```

- b) The body (`i = i - 0.6;`) of the following loop is executed less than 4 times. Yes, No

```
int i = 3.7;
while (i > 0) {
    i = i - 0.6;
}
```

- c) The binary representation of the decimal number 1.625 is equal to 1.1001. Yes, No

- d) Given an arbitrary rational number p/q . There exists a base such that p/q admits a finite representation in the floating point system with this base. Yes, No

e) [Summer 2012, Ex.6]

In this task we are going to write a class `Averager` which computes averages for given inputs. The following program illustrates the functionality by computing the average height of three humans. In general, the average of a sequence of n elements is defined as the sum of these elements divided by n .

```
int main () {
    Averager m;
    m.add_value (1.85);           // human 1
    m.add_value (1.79);           // human 2
    m.add_value (1.64);           // human 3
    std::cout << m.average_value() << "\n"; // Output: 1.76

    return 0;
}
```

In the following program code we are going to write this class `Averager` by defining suitable data members and member functions (Constructor, `add_value`, `average_value`). Also, pay attention to `const`-correctness. Each gap may contain at most one statement. PRE- and POST-conditions are not given here in order to not give you too many hints; however, you shall still assert for dangerous inputs. And you shall also make sure that your specified types do not reduce the precision of any results, but still do not waste memory. And finally, you may leave gaps empty (as long as this does not affect the correctness of the program).

```

// class for computing the average of n values
class Averager {
#1
    unsigned int n; // number of values
    double s;      // sum of values

#2
    Averager () #3 {}

    #4 add_value (#5 v) #6 {
        #7
        #8
    }

    #9 average_value () #10 {
        #11
        #12
    }
};

```

#1: _____
#3: _____
#5: _____
#7: _____
#9: _____
#11: _____

#2: _____
#4: _____
#6: _____
#8: _____
#10: _____
#12: _____