Computer Systems: Object-Oriented Programming in C++

Sample Exam

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

Last Name:

First Name:

Matrikelnummer:

Studienkennzahl:

Please write on the empty space of these sheets; you may also add additional pages. Answers can be written in German or English. All written materials are allowed.

1. (20 points) Write a class `Work` whose objects represent working times (in whole minutes) and salary rates (in whole cents per minute). With this class, the following operation shall be possible:

```
Work* w = new Work(25, 60);  // 25 cent/min, 60 min
w->add(65);                  // add 65 minutes working time
w->printSalary();            // prints salary "31,25" (25*125 Cents)
bool okay = w->subtract(60); // attempts to subtract 60 minutes
                             // returns false, if not sufficient time
                             // available (time remains unchanged)
Work::reset(*w);             // reset working time to zero
Work v(30);                  // 30 cent/min, 0 min
int r = w->compare(v);       // 0 if salaries of w and v are equal,
                             // 1, if w's salary is bigger, -1, else
Work u(v);                   // u becomes a copy of v
```

2. (20 points) Take the following interfaces for an article respectively a shop selling articles:

```
class Article // an article
{
  public:
  virtual string getId() = 0; // its identifier
};

class Shop // a shop
{
  public:
  list<Article*> articles() = 0;
};
```

Write a method

```
set<string> getArticleIds(Shop* s);
```

which takes a shop *s* as parameter and returns as its result the set of the identifiers of all articles contained in *s* (here `set` and `list` represent the templates of the standard library).

3. (25 points) Take the template class

```
template<class T> class BoundedQueue {
  public:
    virtual ~BoundedQueue() { }
    virtual bool isempty() = 0; // is queue empty?
    virtual bool isfull() = 0;  // is queue full?
    virtual void put(T& x) = 0; // add x to queue
    virtual T get() = 0;        // remove element from queue
};
```

which describes the interface of a bounded queue to which (if the queue is not full) elements of type T can be added and from which (if the queue is not empty) elements can be removed (in the order in which they were added). The operations assume that their preconditions (queue is not full/empty) are satisfied.

Write a concrete template class

```
template<class T>
class ArrayQueue: public BoundedQueue<T> { ... };
```

which implements by a constructor

```
ArrayQueue(int s)
```

a bounded queue of size s with the help of an array a, its length l, a counter n (the number of elements in the queue) and two indices h (head) and t (tail): elements are added at position t (t is increased) and removed from position h (h is increased). If h respectively t become s, they are reset to 0 (the technique of *circular buffers*). Actually, t can be determined from h and n and is thus not strictly necessary.

4. (20 points) Consider the template class `List` presented in the course (slide set "Templates", section "Generic Lists"). Write a template function `read` such that the following operation becomes possible:

```
bool isend(double x) { return x == 0; }

List<double> l;
int n = read<double,isend>(l);
```

The function reads a sequence of items $x_1, \ldots, x_{n+1}$ such that $n+1$ is the smallest index i for which `isend`(x_i) returns true; the function puts the elements $x_1, \ldots, x_n$ (in the order in which they were read) into l. The result of the function is n, if everything went okay, and -1, if an error occurred. The function needs time $O(n)$.

You may modify the definition of `List`, e.g. by adding new data members and/or member functions (please indicate your changes clearly); in particular, since new elements are to be added at the end of the list, it is recommended to keep track of its last element.

5. (15 points) Take the declarations

```
class I
{ public: virtual int key() = 0; };
class C: public I
{ public: virtual int key() { ... } int value() { ... } };
class D: public C
{ public: string name() { ... } };
class E
{ public: virtual int key() { ... } };

void print(I* x) { ... }
```

Which of the following declarations/commands yield compilation errors or runtime errors and what is the exact reason?

a) `I* i = new I();`

b) `C* c = new C();`

c) `D* d = new D();`

d) `E* e = new E();`

e) `I* j = c;`

f) `I* k = d;`

g) `I* l = e;`

h) `C* f = d;`

i) `D* g = c;`

j) `D* h = dynamic_cast<D*>(c);`

k) `D* m = dynamic_cast<D*>(f);`

l) `print(j);`

m) `print(c);`

n) `print(d);`

o) `print(e);`

Additionally state explicitly what the values of *h* and *m* are after the assignment (if any).