Computer Systems: Object-Oriented Programming in C++

Sample Exam

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.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)
Work::reset(w);              // reset working time to zero
bool okay = w->subtract(60); // attempts to subtract 60 minutes
                             // returns false, if not sufficient time
                             // available (time remains unchanged)
Work *v = new Work(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) 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 i for which `isend`(x_i) returns true and 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.

You may modify the definition of `List`, e.g. by adding new data members and/or member functions (please indicate your changes clearly).

3. (20 points) Take the following interfaces for an article respectively a shop selling articles:

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

class Shop // a shop
{
  public:

  // number n of articles in shop
  virtual int numberOfArticles() = 0;

  // get article i, 0 <= i < n
  virtual Article* getArticle(int i) = 0;
};
```

Write a method

```
set<Identifier> 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.

4. (25 points) Take the template class

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

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

Write a concrete template class

```
template<class T, unsigned int N>
class ArrayBuffer: public BoundedBuffer<T> { ... };
```

which implements a bounded buffer of size N with the help of an array a, a counter n (the number of elements in the buffer) and two indices f (front) and t (tail): elements are added at position t (t is increased) and removed from position f (f is increased). If t respectively f become N, they are reset to 0 (the technique of *circular buffers*).

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);`