Programming 2:
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:

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
v.reset();                   // reset working time to zero
```

2. (20 points) Take the following interfaces for a music archive:

```
class Song
{
public:
  virtual string title() = 0; // the title of the song
  virtual string interpreter() = 0; // its interpreter
  virtual int length() = 0;   // its length (in seconds)
};
class Album
{
public:
  virtual int year() = 0;        // the year of the album
  virtual string title() = 0;    // the title of the album
  virtual int size() = 0;        // the number of songs in the album
  virtual Song* song(int i) = 0; // song i, 0 <= i < size()
};
class Archive
{
public:
  virtual int size() = 0;          // the number of albums in the archive
  virtual Album* album(int i) = 0; // album i, 0 <= i < size()
};
```

Write a function

```
int printSongs(Archive* archive, int year, string interpreter)
```

which takes an archive *archive*, an integer *year*, and a string *interpreter*. It prints to the standard output stream in a line of the form

```
albumTitle: songTitle
```

every song performed by *interpreter* on every album in *archive* published in *year*. The function returns the sum of the lengths of these songs.

Hint: iterate over every *album* of the archive and; if it was not published in the denoted year, ignore it. Otherwise, call a (self-defined) auxiliary function `printAlbum(album, interpreter)` that prints the titles of all songs from *album* published by *interpreter* and returns the sum of the lengths of these songs.

This auxiliary function must be defined and used.

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. (25 points) Implement a function with the following header (where `map`, `set`, and `list` are class templates of the C++ standard library):

   ```
   map< string, set<int> > occurrences(list<string> text)
   ```

   This function traverses a text (a list of words) and returns as a result a map that assigns to every word of the text the set of all positions where the word occurs in the text. For instance, if the text consists of the words

   ```
   der die das die der der
   0   1   2   3   4   5
   ```

   then the result maps "der" to the set $\{0, 4, 5\}$, "die" to $\{1, 3\}$ and "das" to $\{2\}$.

   For this, first generate an empty map. Then, for every word in the text, check whether the map has already an entry for the word. If yes, add the position of the word to the corresponding set; if not, create a new set with the single position. In any case, be sure that the map indeed contains the (new/updated) set.

5. (10 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).