CITY
UNIVERSITY OF LONDON
EST 1894

School of Science & Technology

Department of Computer Science

BSc/MSci Computer Science
BSc/MSci Computer Science with Computer Games Technology
MSci Computer Science with Cyber Security
MSci Data Science

IN2029: Programming in C++

Stage 2 Examination

January 2024

Examination duration: **90 minutes**

Division of marks: Full marks may be obtained for correct answers to **both** of the **Two** questions.

Other instructions:

BEGIN EACH QUESTION ON A FRESH PAGE

Number of answer books to be provided: 1
Calculators permitted: Sharp EL-531TS/ EL-531TH/ EL-531TG or Casio FX-83/85 MS/ES/GT+/GTX ONLY
Examination duration: 90 minutes
Dictionaries permitted: None
Can question paper be removed from the examination room: No
Additional materials: None

External Examiner:	Prof. Stuart Anderson
Internal Examiner:	Dr Ross Paterson

1. (a) Write a function that takes a list of doubles, and modifies the list by multiplying each element by 2. **[5 Marks]**

   (b) Using the following library algorithms:

   find(b, e, v) searches the range $[b, e)$ and returns an iterator pointing at the first occurrence of v, if there is one, or e if there is not.

   count(b, e, v) returns the number of elements in the range $[b, e)$ that are equal to v.

   (The notation $[b, e)$ denotes a range starting at the position of the iterator b and extending up to but not including the position of the iterator e.)

   (i) Write a function that takes a list of doubles and returns the number of zeroes in the list. **[5 Marks]**

   (ii) Write a function that takes a list of doubles and returns the number of zeroes before the first 1 in the list. **[10 Marks]**

   (iii) Write a function that takes a list of doubles and replaces the first 1 (if any) by 0. **[10 Marks]**

   (c) (i) Write an abstract base class `loggable` with a member function `log` that takes a string and a number. Implementations in derived classes (which you are not asked to write) might print a message on the console, add an entry to a container or a database, or something else. They will not, however, modify the string. **[10 Marks]**

   (ii) Explain, with a code fragment, how you would implement an arbitrary-sized collection of objects of different concrete derived classes of `loggable`, including how you would ensure that their memory was recycled when no longer required. **[10 Marks]**

2. (a) Define a class `item` for valuable items in a game. Each item should have a description (a string) and a value (a whole number). Include a constructor and appropriate accessor member functions. **[15 Marks]**

   (b) Given the library algorithm

   sort(b, e, f) Sort the elements in the range $[b, e)$ using the function f as a less-than function on values, i.e. $f(x, y)$ returns `true` if x should be placed before y.

   write an independent function that takes a vector of `item` objects and sorts them in order of descending value (so that the most valuable items come first and the least valuable last). **[10 Marks]**

   (c) Define a `sack` class containing an arbitrary number of `item` records. Your class should include member functions to

   (i) add a new `item`.

   (ii) return the total value of all the items in the sack.

   (iii) given a description string, check whether the sack contains an item with that description.

   [15 Marks]

   (d) Using the following library algorithm:

   count_if(b, e, p) returns the number of elements in the range $[b, e)$ for which p is `true`.

   write a member function of the `sack` class that takes as a parameter a number `n` and returns the number of items in the sack with a value greater than `n`. **[10 Marks]**

End of paper

Marking Scheme

1. (a) 5 marks

```
void double_list(list<double> &l) {
    for (auto &x : l)
        x *= 2;
```

Marking: signature [2, including 1 for the ref], loop structure [2, including 1 for the ref], loop body [1].

An explicit iterator is equally acceptable:

```
void double_list(list<double> &l) {
    for (list<double>::iterator p = l.begin(); p != l.end(); ++
        *p *= 2;
}
```

(b) (i) 5 marks

```
int count_zeroes(const list<double> &l) {
    return count(l.cbegin(), l.cend(), 0);
}
```

Marking: signature [3, including 2 for const ref], body [2].

(ii) 10 marks

```
int count_zeroes_before_one(const list<double> &l) {
    list<double>::const_iterator p =
        find(l.cbegin(), l.cend(), 1);
    return count(l.cbegin(), p, 0);
}
```

Marking: signature [3, including 2 for const ref], find [3], count [4].

(iii) 10 marks

```
void replace_one(list<double> &l) {
    list<double>::iterator p = find(l.begin(), l.end(), 1);
    if (p != l.end())
        *p = 0;
}
```

Marking: signature [3, including 2 for ref], find [2], test [3], assignment [2].

(c) (i) Expect something like: 10 marks

```
class loggable {
public:
    virtual ~loggable(){}
    virtual void log(const string &s, int n) = 0;
};
```

Marking: virtual destructor [2], pure virtual function [6], argument types [2].

(ii) 2 marks for an invalid answer like **10 marks**

```
vector<loggable> loggers;
```

because we cannot create `loggable` objects.
Full marks for an answer using shared pointers:

```
vector<shared_ptr<loggable>> loggers;
```

When the container is destroyed, the shared pointers will also be destroyed, and if the reference counts fall to 0, so will the objects. `unique_ptr` would also be fine here.
An answer involving pointers

```
vector<loggable *> loggers;
```

gets 6 marks, plus 2 more if they say they will use `delete` on each element of the container when finished.

2. (a) 15 marks

```
class item {
    string _desc;
    int _value;

public:
    item(const string & d, int v) :
        _desc(d), _value(v) {}

    const string & desc() const { return _desc; }

    int value() const { return _value; }
};
```

Marking: members [2], constructor [6] (2 for const and ref), accessors [7] (2 for const and ref). Additional const and ref for the `int` are optional.

(b) 10 marks

```
bool compare(const item &x, const item &y) {
    return x.value() > y.value();
}

void sort_items(vector<item> &v) {
    sort(v.begin(), v.end(), compare);
}
```

A lambda function is also acceptable.

Marking: compare function: signature [3, including 2 for const ref], body [2]. sort: signature [3, including 2 for ref], body [2].

(c) 15 marks

```
class sack {
    vector<item> items;

public:
    void add_entry(const item & entry) {
        items.push_back(entry);
    }

    int total_value() const {
        int total = 0;
        for (const item & entry : items)
            total += entry.value();
        return total;
    }

    bool search(const string & d) const {
        for (const item & entry : items)
            if (entry.desc() == d)
                return true;
        return false;
    }
};
```

No constructor is necessary; the compiler-generated default constructor initializes an empty vector of items. Indexing loops or range-based loops are acceptable in place of iterator loops. Any sequence type will do in place of `vector`, but if they use `list`, they must use iterators (or range-based loops). They can even use a `map` indexed by item name.

Marking: class [1], field [1], add [3, incl. 2 for const ref], total value [5, incl. 1 for const], search [5, incl. 1 for const, 2 for const ref]. Versions that correctly maintain the total in another data member are also acceptable.

(d) This requires a lambda function: 10 marks

```
    int count_valuable(int n) const {
        return count_if(items.cbegin(), items.cend(),
            [n] (const entry  &i) { return i.value() > n; });
    }
```

Marking: signature [2] (including 1 for `const`), algorithm call [3], lambda function [5].