

CS 14: Data Structures and Algorithms

Quiz 0

Wagner Truppel
wagner@cs.ucr.edu

Peter H. Fröhlich
phf@cs.ucr.edu

January 7, 2003

Time: 30 Minutes

Start here: Please fill in the following important information using a **permanent** pen **before** you do **anything** else! Your exam will **not** be graded if you use a pencil or erasable ink on this page.

Name (print): _____

Student ID (print): _____

Signature: _____

Now have **two** neighbors **confirm** your identity by **signing** below. Provide **official** identification (your UCR card, driver's license, etc.) if requested. Your exam will **not** be graded without these.

Neighbor 1: _____

Neighbor 2: _____

Instructions: Read these instructions carefully before you start. This quiz is intended primarily to give the instructors a sense for where the class stands. It is **not** worth a significant portion of your grade. Read each question carefully and answer to the best of your ability. And please don't **guess**, in this quiz it's better not to answer than to answer by guessing!

Switch off your phones, pagers, and other noisy gadgets! You are **not** allowed to have anything but a pen and this exam on your desk. You are **not** allowed to talk to anyone during the exam. If you have a question, raise your hand **quietly** and someone will assist you. You have to **remain seated quietly** until we have collected all the exams. Remember that you can **not** claim grading errors if you do not use a **permanent** pen for your answers.

Do not open before you are told to do so!

Score: _____ **out of 25 points.**

1 Warmup (5 points)

Determine whether the following statements are **true** or **false**. (1 point each)

1. The expressions “`a[i++]`” and “`a[++i]`” access the **same** array element (assume `int a[10]` and `int i = 0`).

False. The first accesses `a[0]`, the second accesses `a[1]`.

2. The expressions “`i = i + 1`”, “`i += 1`”, and “`++i`” **increase** the value of `i` (assume `int i`; if false, state your reason in one word).

True (most of the time anyway). False in case of **overflow**.

3. An algorithm A which requires $O(n \log n)$ time is **always** faster than an algorithm B which requires $O(n^2)$ time.

False. For small n algorithm B might be faster.

4. Recursion (in algorithms) is **related** to induction (in mathematics).

True. In some sense they are actually identical; we often prove the correctness of an recursive algorithm by induction.

5. In the 1930s, Los Angeles had one of the largest **public** transportation systems in the United States.

True. Such systems were destroyed systematically across the US by the emerging car and oil industry. Nice.

2 Definitions (4 points)

Define or briefly explain each of the following terms in one sentence. (1 point each)

1. Inheritance:

Incremental modification mechanism that allows new classes to be created from old ones through additions of variables and methods; often tied to subtyping, resulting in a number of problems.

2. Polymorphism:

Literally “many forms,” in programming most often the ability to define operations of the same name applicable to objects of different types.

3. Abstract base class:

A class that defines an interface without providing an implementation; cannot be instantiated.

4. Constructor:

A special method executed when an object is created to initialize the object (possibly using certain parameters).

3 General Topics (12 points)

1. **Protection** of member variables and member functions in classes for C++. (5 points)

(a) What are the three levels of protection provided?

Public, protected, private.

(b) Describe each in one sentence, focusing on the differences between them.

Public = accessible from outside the class, protected = accessible from inside the class and from all subclasses, private = accessible from inside the class only; friend classes can break all of these.

2. **Operator overloading** in C++. (4 points)

(a) Define or briefly explain operator overloading?

The same operators (e.g. `operator+`) that are used for built-in types (e.g. `int`) can be defined for user-defined types as well.

(b) Why would you use it? Explain or provide an example.

In a library supporting various algebraic structures, operator overloading would be used to provide the different types (e.g. `complex`, `vector`, `matrix`) with a familiar (from mathematics) set of operators. This can make the abstractions easier to use.

4 Advanced Topics (4 points)

1. **Templates** in C++. (2 points)

(a) Define or briefly explain a `template` and why you might use one.

Templates allow parameterization by types instead of just values (as e.g. functions do). This is useful when the exact type to be used should be kept open until compile time. Also, most ADTs perform the same operations regardless of the type of data they are actually operating on—for instance a stack always supports `push()` and `pop()`—regardless of what type of object is being managed. The STL (standard template library) provides a wealth of ADTs defined as templates, which can operate on a wide range of suitable objects.

(b) What is the C++ syntax for instantiating a variable named `vec` as an instance of a `vector` template containing `ints`?

```
vector<int> vec;
```

2. Consider the following C++ code snippet (you might have to make certain assumptions not explicitly stated): (2 points)

```
const int MAX = 10;

int foo(int bar) {
    cout << "input was: " << bar << endl;
    if ( bar > MAX ) {
        out_of_range e("input out of range");
        throw e;
    }
    return bar++;
}

int main ( int argc, char **argv ) {
    int input = atoi(argv[1]);
    int output = input;
    try {
        output = foo (input);
        cout << "output is: " << output << endl;
    } catch ( exception &e ) {
        cout << "exception caught: " << e.what() << endl;
    }
}
```

What is the program output if the input is:

- (a) 5

```
input was: 5
output is: 5
```

If this isn't intuitive, notice that the instruction `return bar++` contains a *post*-increment—meaning that the value is not incremented until **after** the return executes.

- (b) 10

```
input was: 10
output is: 10
```

- (c) 15

```
input was: 15
exception caught: input out of range
```

- (d) not provided (i.e. no arguments are given)

Undefined/anything, since illegal array reference. If you're lucky, you'll probably see `Segmentation fault` under Unix.