CS 32 Fall '21 Midterm 2

Answer Key

Instructions

- Please read all the instructions very carefully.
- This exam is graded out of 55 points.
- Make sure that the handwriting in your solutions is legible.
- **For questions 2–6**, you can write *"I don't know but I promise I will learn"* and get 20% of the points.
- One double-sided cheat sheet is allowed, you need to turn it in with the exam itself.
- Course materials or computing devices are **not** allowed during the exam.
- Write your name and PERM # on top of each page.

1. (10 points) Total for Question 1: 10

   Mark whether each statement is True or False. If **false**, *briefly* state why.

   1. (T | F) Only one of the `catch` blocks attached to a `try` block can be executed when an exception is thrown from inside of the `try` block.
      True.
   2. (T | F) Complete testing is always feasible.
      False. It is not feasible when the input space is too large (all doubles), or infinite (all linked lists).
   3. (T | F) In C++, any object can be thrown using a `throw` statement.
      True.
   4. (T | F) Dynamic dispatch is used for all methods of a class.
      False. It is used only for virtual methods.
   5. (T | F) If the destructor is virtual, then the constructor also has to be virtual.
      False. Constructors cannot be virtual.

2. Total for Question 2: 12

(a) (6 points) In the snippet below, write down the output after each statement in `main` to the space provided right next to it.

```
struct A {
  virtual void call() { cout << "called A\n"; }
  virtual ~A() = default;
};
struct B : public A {
  void call() { cout << "called B\n"; }
  void call() const { cout << "called B const\n"; }
};
struct C : public B {
  void call() override { cout << "called C\n"; }
};

void f(A a) { a.call(); }
void g(A& a) { a.call(); }

int main(int argc, char *argv[]) {
  A a; B b; C c;
  f(a);                       // called A
  f(b);                       // called A
  f(c);                       // called A
  g(a);                       // called A
  g(b);                       // called B
  g(c);                       // called C
  return 0;
}
```

(b) (3 points) In the code above, object slicing may happen in some calls inside `main`, write down below the calls in which object slicing happens:

Solution: f(b) and f(c).
Note: Slicing doesn't happen on f(a) because type of a is `A` already.

(c) (3 points) In the code snippet above, there are two definitions (overloads) of `call` for `B` but only one of them provides an alternative to (overrides) the virtual function defined in `A`. What is the keyword in C++ to make the compiler check that a definition overrides a virtual function from a parent class? Why is that keyword useful?

Solution: The keyword is `override`. It is useful to make compiler enforce that we are overriding a parent class method, rather than defining a new virtual (or non-virtual) method with a similar but different signature *by accident*.

3. (10 points) Given the code below, write the output after each statement in the main function is executed.

```
using namespace std;

class Foo {};
class Bar {};
class Baz : public Foo {};
class Quux : public Baz {};

void f(int x) {
  try {
    switch (x % 4) {
      case 1: throw Foo{};
      case 2: throw Bar{};
      case 3: throw Baz{};
      case 0: throw Quux{};
    }
  } catch (Baz&)  { cout << "Caught Baz"  << endl; }
    catch (Bar&)  { cout << "Caught Bar"  << endl; }
    catch (Quux&) { cout << "Caught Quux" << endl; }
    catch (Foo&)  { cout << "Caught Foo"  << endl; }
}

int main(int argc, char * argv) {
  f(0);                          // Caught Baz
  f(1);                          // Caught Foo
  f(2);                          // Caught Bar
  f(3);                          // Caught Baz
  f(4);                          // Caught Baz
  return 0;
}
```

4. Implement a function `Max` that returns the maximum value in a vector. Since calling `Max` on an empty vector does not make sense, you need to handle that case by *returning an optional value*. `Max` should accept a `vector<double>` input, and return an optional `double`. You can assume that the necessary headers are included, and `using namespace std;` already appears in the current scope.

Total for Question 4: 6

(a) (2 points) Give the signature (prototype) of `Max` below. Since it doesn't throw an exception, mark it as such using `noexcept`:

```
optional<double> Max(const vector<double> & input) noexcept;
```

(b) (4 points) Complete the implementation of `Max` below (the parts marked `/*...*/` in the signature are deliberately omitted as they are the answer to the previous part):

```
/* ... */ Max(/* ... */ input) /* ... */ {
  if (input.empty()) {
    return nullopt; // the first blank
  }
  double current_max = input[0];
  for (double x : input) if (x > current_max) current_max = x;
  return current_max; // the second blank
}
```

5. Total for Question 5: 8

(a) (4 points) Why should we (as software engineers) care about automated testing? To expand, what is the benefit of going through and setting up an automatic way of running tests?

Solution: An automated test suite makes it easier to check for the correctness of our code. So, we can check if we reintroduce a bug (a regression), or create a bug because a test starts failing. Automated testing allow us to integrate testing into the software engineering workflow to make these checks automatically, similar to how the autograders in Gradescope are run on each submission.

(b) (4 points) What is *mocking*? Where is it used? Briefly explain.

Solution: Mocking is creating an implementation of a class that *mocks (mimics)* the behavior of the classes we depend on for the limited cases we need in testing, so we can replace our dependencies during testing. It is used in unit testing, so that our unit tests' results do not depend on other classes implementation.

6. A useful generic function is `ArgMax`: it returns an element of a collection that maximizes an objective function, and it is used in the formulation of many optimization problems. For example, we can use it to find the longest string in a list (the string that maximizes the "length" function). You will implement a version of `ArgMax` that takes two arguments a `vector<T>` input, and a function `f` from `T` to `int` (that is, `f` itself takes a single argument of type `T` and returns an `int`).

Total for Question 6: 9

(a) (2 points) Complete the signature of `ArgMax` below by filling in the second argument `f`:

```
template<typename T>
T ArgMax(vector<T> input, _ function<double(int)> f _);
```

(b) (3 points) Complete the implementation of `ArgMax` using a for loop below:

```
template<typename T>
T ArgMax(vector<T> input, /* ... */ f) {
  T arg_max_so_far = input.at(0);
  for (const auto& x : input) {
    if (_ f(arg_max_so_far) < f(x) _) {}
      arg_max_so_far = _ x _;
    }
  }
  return arg_max_so_far;
}
```

(c) (4 points) Implement `ArgMax` using `transform` and `reduce` (or just `reduce` (you don't need `transform`, strictly speaking), or using `transform_reduce` if you want). You can assume (like the case above) that `input` has at least one element. You can define helper functions or lambdas (anonymous functions) if you need any.

Solution: (ignoring using references for efficiency):

```
T ArgMax(vector<T> input, function<double(T)> f) {
  return reduce(input.begin(), input.end(), input[0],
                [f](T a, T b) {
                  if (f(a) > f(b)) return a;
                  return b;
                });
}
```

In the answer above, the lambda needs to capture `f` so it can use it, so we write it in brackets. Because we did not cover this in the lectures, you won't get any points deducted for that.

Scratch Paper